Load The Dataset (Week 2)

In [1]:
```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

#ingest data
df = pd.read_csv('https://raw.githubusercontent.com/Christine971224/Analytics-2023/maste
df.head()
```

Out[1]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival
0	Resort Hotel	0	342	2015	July	27	
1	Resort Hotel	0	737	2015	July	27	
2	Resort Hotel	0	7	2015	July	27	
3	Resort Hotel	0	13	2015	July	27	
4	Resort Hotel	0	14	2015	July	27	

5 rows × 36 columns

In [2]:
```
#basic information of dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 119390 entries, 0 to 119389
Data columns (total 36 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   hotel                           119390 non-null  object 
 1   is_canceled                     119390 non-null  int64  
 2   lead_time                       119390 non-null  int64  
 3   arrival_date_year               119390 non-null  int64  
 4   arrival_date_month              119390 non-null  object 
 5   arrival_date_week_number        119390 non-null  int64  
 6   arrival_date_day_of_month       119390 non-null  int64  
 7   stays_in_weekend_nights         119390 non-null  int64  
 8   stays_in_week_nights            119390 non-null  int64  
 9   adults                          119390 non-null  int64  
 10  children                        119386 non-null  float64
```

```
 11  babies                          119390 non-null  int64
 12  meal                            119390 non-null  object
 13  country                         118902 non-null  object
 14  market_segment                  119390 non-null  object
 15  distribution_channel            119390 non-null  object
 16  is_repeated_guest               119390 non-null  int64
 17  previous_cancellations          119390 non-null  int64
 18  previous_bookings_not_canceled  119390 non-null  int64
 19  reserved_room_type              119390 non-null  object
 20  assigned_room_type              119390 non-null  object
 21  booking_changes                 119390 non-null  int64
 22  deposit_type                    119390 non-null  object
 23  agent                           103050 non-null  float64
 24  company                         6797 non-null    float64
 25  days_in_waiting_list            119390 non-null  int64
 26  customer_type                   119390 non-null  object
 27  adr                             119390 non-null  float64
 28  required_car_parking_spaces     119390 non-null  int64
 29  total_of_special_requests       119390 non-null  int64
 30  reservation_status              119390 non-null  object
 31  reservation_status_date         119390 non-null  object
 32  name                            119390 non-null  object
 33  email                           119390 non-null  object
 34  phone-number                    119390 non-null  object
 35  credit_card                     119390 non-null  object
dtypes: float64(4), int64(16), object(16)
memory usage: 32.8+ MB
```

In [3]:

```
df.isnull().mean()
```

Out[3]:

```
hotel                             0.000000
is_canceled                       0.000000
lead_time                         0.000000
arrival_date_year                 0.000000
arrival_date_month                0.000000
arrival_date_week_number          0.000000
arrival_date_day_of_month         0.000000
stays_in_weekend_nights           0.000000
stays_in_week_nights              0.000000
adults                            0.000000
children                          0.000034
babies                            0.000000
meal                              0.000000
country                           0.004087
market_segment                    0.000000
distribution_channel              0.000000
is_repeated_guest                 0.000000
previous_cancellations            0.000000
previous_bookings_not_canceled    0.000000
reserved_room_type                0.000000
assigned_room_type                0.000000
booking_changes                   0.000000
deposit_type                      0.000000
agent                             0.136862
company                           0.943069
days_in_waiting_list              0.000000
customer_type                     0.000000
adr                               0.000000
required_car_parking_spaces       0.000000
```

```
total_of_special_requests        0.000000
reservation_status               0.000000
reservation_status_date          0.000000
name                             0.000000
email                            0.000000
phone-number                     0.000000
credit_card                      0.000000
dtype: float64
```

In [4]:

```
# adults, babies and children can't be zero at same time, so dropping the rows having a
filter = (df.children == 0) & (df.adults == 0) & (df.babies == 0)
df[filter]
```

Out[4]:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number
2224	Resort Hotel	0	1	2015	October	41
2409	Resort Hotel	0	0	2015	October	42
3181	Resort Hotel	0	36	2015	November	47
3684	Resort Hotel	0	165	2015	December	53
3708	Resort Hotel	0	165	2015	December	53
...	...	...	...	...	...	...
115029	City Hotel	0	107	2017	June	26
115091	City Hotel	0	1	2017	June	26
116251	City Hotel	0	44	2017	July	28
116534	City Hotel	0	2	2017	July	28
117087	City Hotel	0	170	2017	July	30

180 rows × 36 columns

In [5]:

```python
# transpose the resulting DataFrame
df.describe([0.01,0.05,0.1,0.25,0.5,0.75,0.99]).T
```

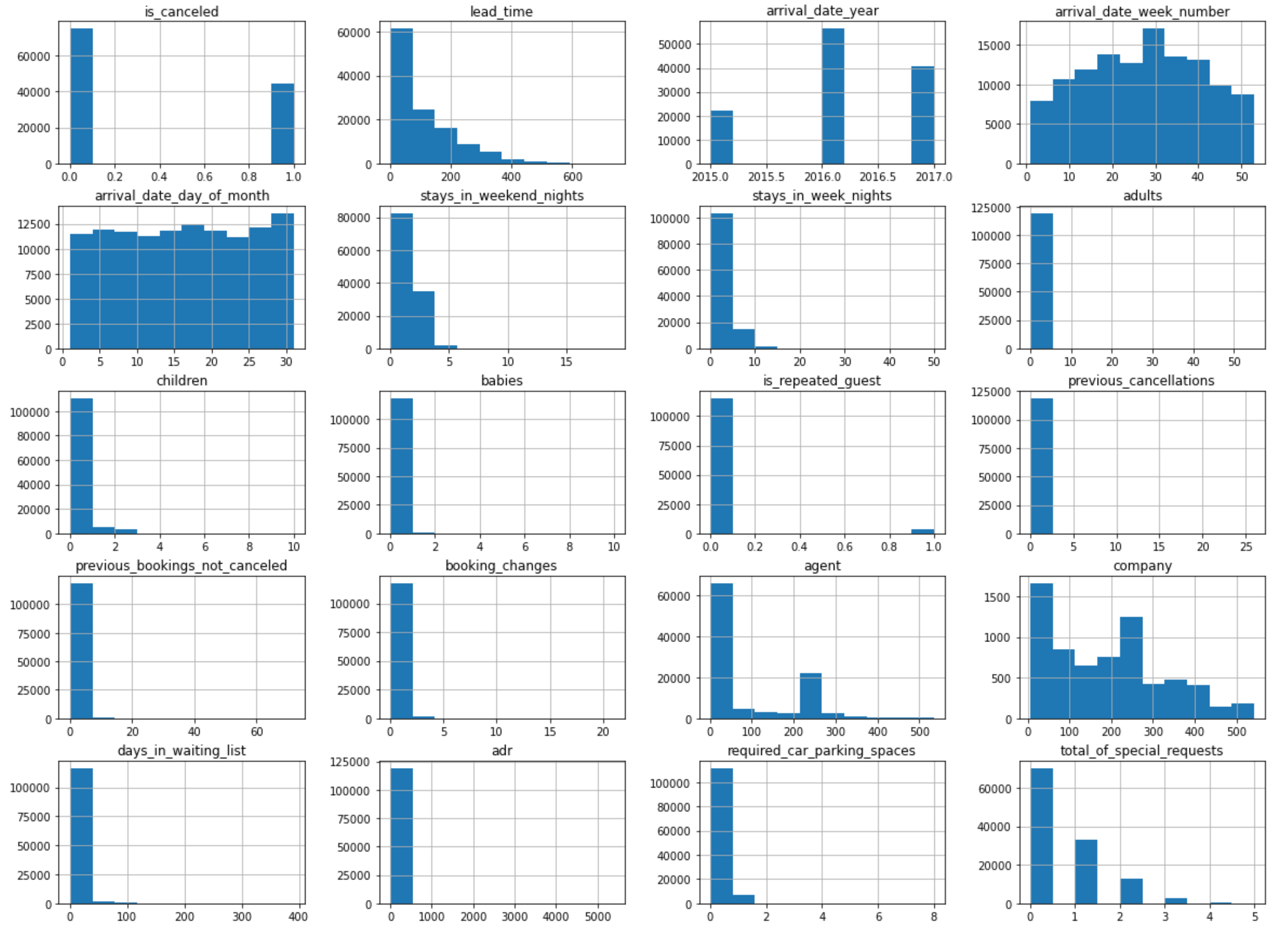
Out[5]:

	count	mean	std	min	1%	5%	10%	25
is_canceled	119390.0	0.370416	0.482918	0.00	0.0	0.0	0.0	0.
lead_time	119390.0	104.011416	106.863097	0.00	0.0	0.0	3.0	18.
arrival_date_year	119390.0	2016.156554	0.707476	2015.00	2015.0	2015.0	2015.0	2016.
arrival_date_week_number	119390.0	27.165173	13.605138	1.00	2.0	5.0	8.0	16.
arrival_date_day_of_month	119390.0	15.798241	8.780829	1.00	1.0	2.0	4.0	8.
stays_in_weekend_nights	119390.0	0.927599	0.998613	0.00	0.0	0.0	0.0	0.
stays_in_week_nights	119390.0	2.500302	1.908286	0.00	0.0	0.0	1.0	1.
adults	119390.0	1.856403	0.579261	0.00	1.0	1.0	1.0	2.
children	119386.0	0.103890	0.398561	0.00	0.0	0.0	0.0	0.
babies	119390.0	0.007949	0.097436	0.00	0.0	0.0	0.0	0.
is_repeated_guest	119390.0	0.031912	0.175767	0.00	0.0	0.0	0.0	0.
previous_cancellations	119390.0	0.087118	0.844336	0.00	0.0	0.0	0.0	0.
previous_bookings_not_canceled	119390.0	0.137097	1.497437	0.00	0.0	0.0	0.0	0.
booking_changes	119390.0	0.221124	0.652306	0.00	0.0	0.0	0.0	0.
agent	103050.0	86.693382	110.774548	1.00	1.0	1.0	6.0	9.
company	6797.0	189.266735	131.655015	6.00	16.0	40.0	40.0	62.
days_in_waiting_list	119390.0	2.321149	17.594721	0.00	0.0	0.0	0.0	0.
adr	119390.0	101.831122	50.535790	-6.38	0.0	38.4	50.0	69.
required_car_parking_spaces	119390.0	0.062518	0.245291	0.00	0.0	0.0	0.0	0.
total_of_special_requests	119390.0	0.571363	0.792798	0.00	0.0	0.0	0.0	0.

In [6]:

```python
import matplotlib.pyplot as plt

# generate histograms for all the columns
df.hist(figsize=(20,15))
plt.show()
```



EDA（Week 3）

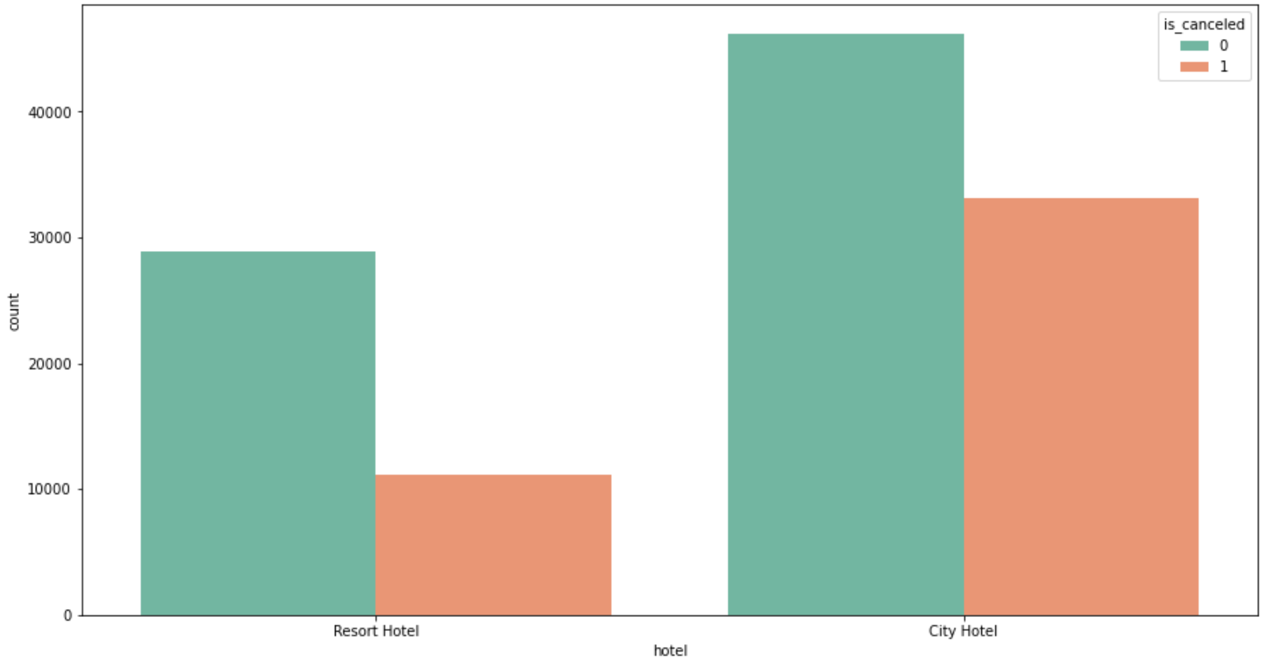
1. Hotel bookings and cancellations

In [7]:

```
# The number of hotel reservations and cancellations can directly show the actual numbe
import seaborn as sns
plt.figure(figsize=(15,8))
sns.countplot(x='hotel'
             ,data=df
             ,hue='is_canceled'
             ,palette=sns.color_palette('Set2',2)
            )
```

Out[7]:

```
<AxesSubplot:xlabel='hotel', ylabel='count'>
```



In [8]:

```
#calculate the proportion of cancellations for each unique value in the 'hotel' column (
hotel_cancel=(df.loc[df['is_canceled']==1]['hotel'].value_counts()/df['hotel'].value_co
print('Hotel cancellations'.center(20),hotel_cancel,sep='\n')
```

```
Hotel cancellations
City Hotel      0.417270
Resort Hotel    0.277634
Name: hotel, dtype: float64
```

Comment: City Hotel's booking volume and cancellation volume are both higher than Resort Hotel's, but Resort Hotel's cancellation rate is 27.8%, while City Hotel's cancellation rate reaches 41.7%.

1. Hotel bookings by month

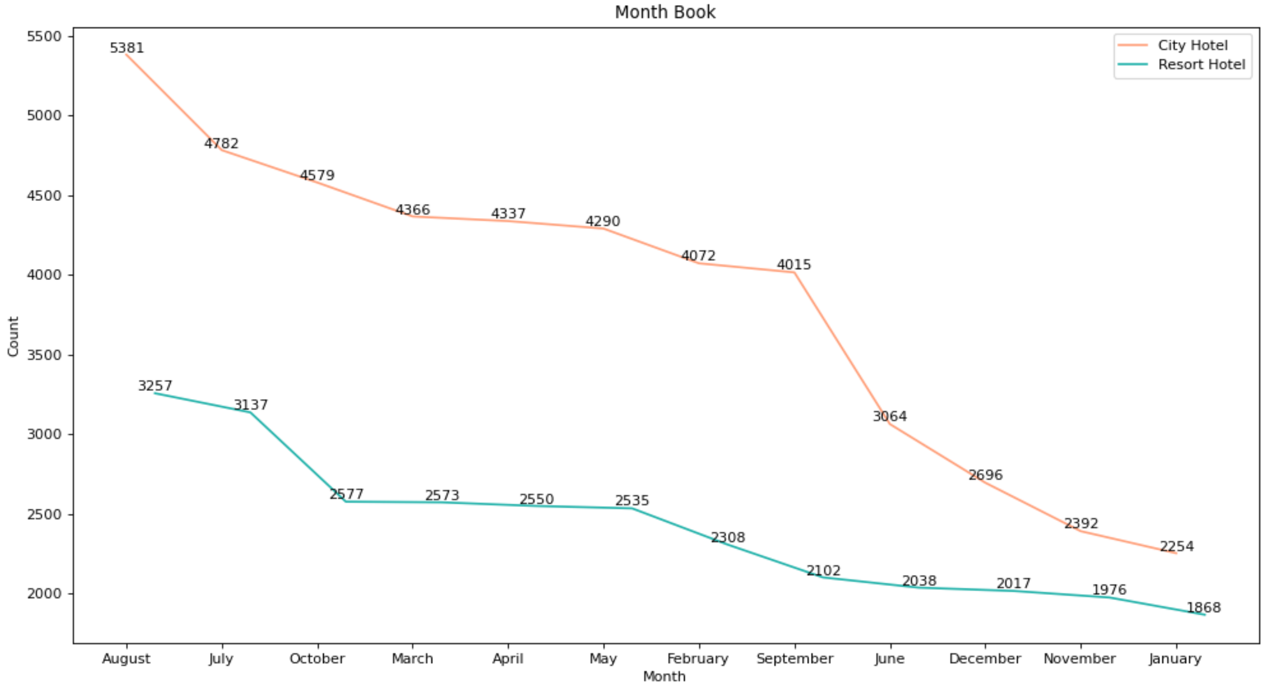
In [9]:

```
#create a plot to visualize the number of bookings for "City Hotel" and "Resort Hotel" (
city_hotel=df[(df['hotel']=='City Hotel') & (df['is_canceled']==0)]
resort_hotel=df[(df['hotel']=='Resort Hotel') & (df['is_canceled']==0)]
for i in [city_hotel,resort_hotel]:
    i.index=range(i.shape[0])

city_month=city_hotel['arrival_date_month'].value_counts()
resort_month=resort_hotel['arrival_date_month'].value_counts()
name=resort_month.index
x=list(range(len(city_month.index)))
y=city_month.values
x1=[i+0.3 for i in x]
y1=resort_month.values
width=0.3
plt.figure(figsize=(15,8),dpi=80)
plt.plot(x,y,label='City Hotel',color='lightsalmon')
plt.plot(x1,y1,label='Resort Hotel',color='lightseagreen')
plt.xticks(x,name)
plt.legend()
plt.xlabel('Month')
plt.ylabel('Count')
plt.title('Month Book')
```

```
for x,y in zip(x,y):
    plt.text(x,y+0.1,'%d' % y,ha = 'center',va = 'bottom')

for x,y in zip(x1,y1):
    plt.text(x,y+0.1,'%d' % y,ha = 'center',va = 'bottom')
```

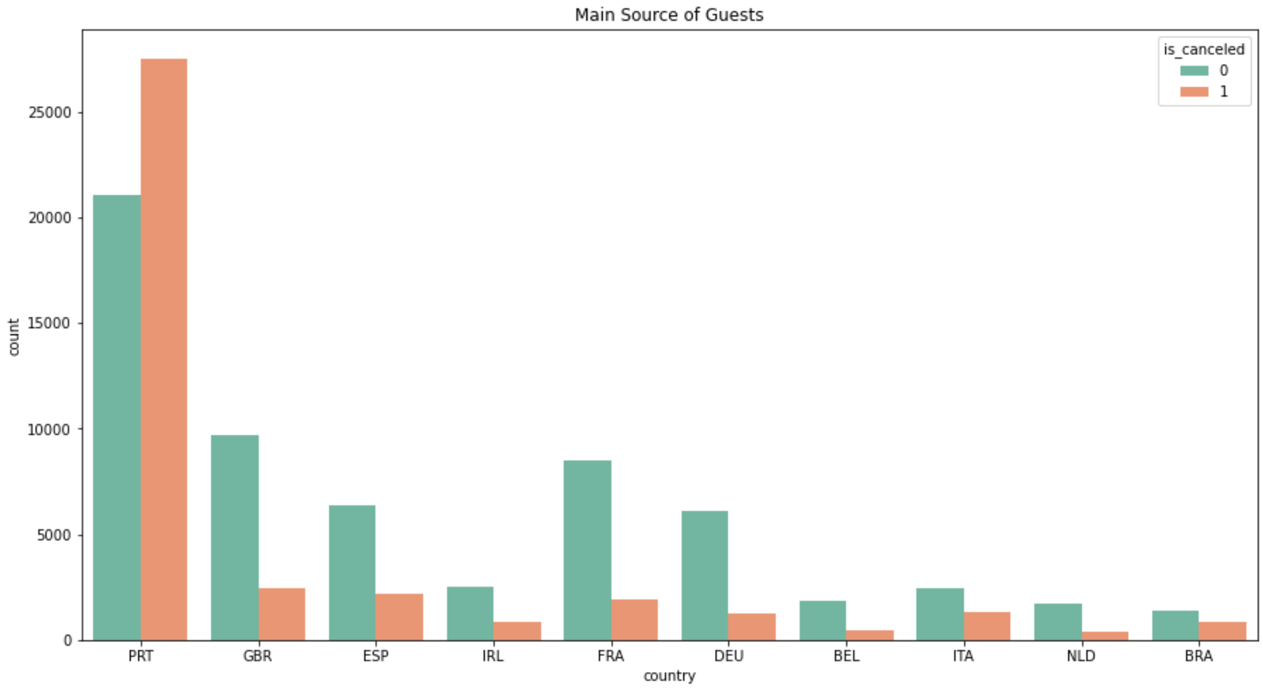


Comment: Peak booking months are August and July. Preliminary judgment is that the long holiday caused the peak period.

1. Customer origin and booking cancellation rate

In [10]:
```
#create a plot using Seaborn's countplot to visualize the top 10 countries from which bo
country_book=df['country'].value_counts()[:10]
country_cancel=df[(df.country.isin (country_book.index)) & (df.is_canceled==1)]['countr
plt.figure(figsize=(15,8))
sns.countplot(x='country'
              ,data=df[df.country.isin (country_book.index)]
              ,hue='is_canceled'
              ,palette=sns.color_palette('Set2',2)
             )
plt.title('Main Source of Guests')
```

Out[10]:
```
Text(0.5, 1.0, 'Main Source of Guests')
```



In [11]:

```
#calculate the cancellation rate for each of the top 10 countries (those with the highe
country_cancel_rate=(country_cancel/country_book).sort_values(ascending=False)
print('Customer cancellation rates by country'.center(10),country_cancel_rate,sep='\n')
```

```
Customer cancellation rates by country
PRT    0.566351
BRA    0.373201
ITA    0.353956
ESP    0.254085
IRL    0.246519
BEL    0.202391
GBR    0.202243
FRA    0.185694
NLD    0.183935
DEU    0.167147
Name: country, dtype: float64
```

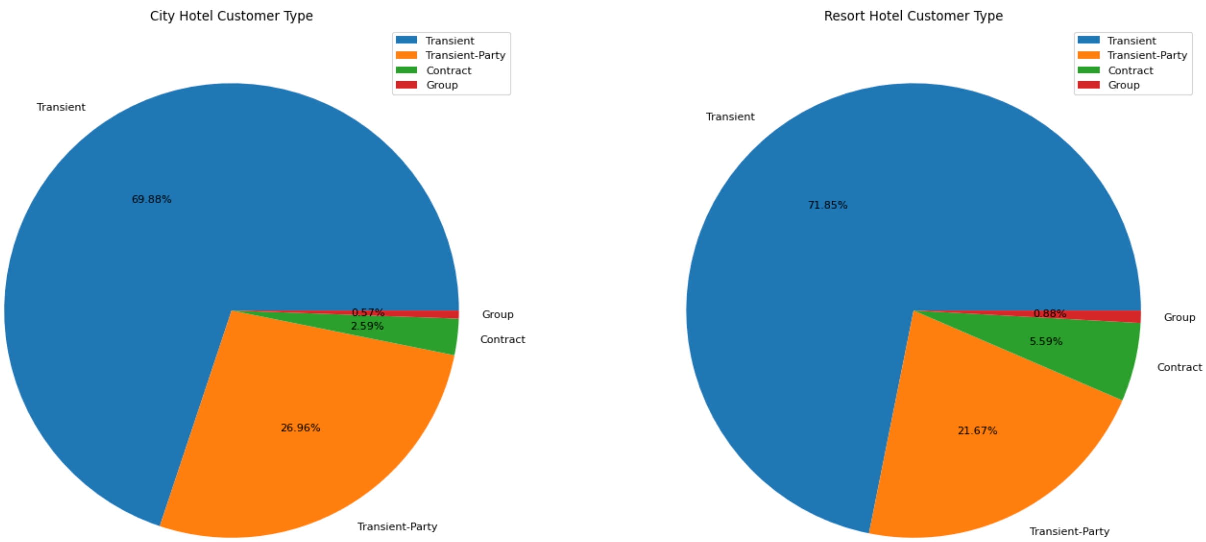
The peak season for both Resort hotel and City hotel is July and August in summer, and the main sources of tourists are European countries. This is in line with the characteristics of European tourists who prefer summer travel. It is necessary to focus on countries with high cancellation rates such as Portugal (PRT) and the United Kingdom (BRT). Main source of customers.

1. Customer type

In [12]:

```
#visualize the distribution of customer types for two types of hotels: City Hotel and Re
city_customer=city_hotel.customer_type.value_counts()
resort_customer=resort_hotel.customer_type.value_counts()
plt.figure(figsize=(21,12),dpi=80)
plt.subplot(1,2,1)
plt.pie(city_customer,labels=city_customer.index,autopct='%.2f%%')
plt.legend(loc=1)
plt.title('City Hotel Customer Type')
plt.subplot(1,2,2)
plt.pie(resort_customer,labels=resort_customer.index,autopct='%.2f%%')
```

```
plt.title('Resort Hotel Customer Type')
plt.legend()
plt.show()
```

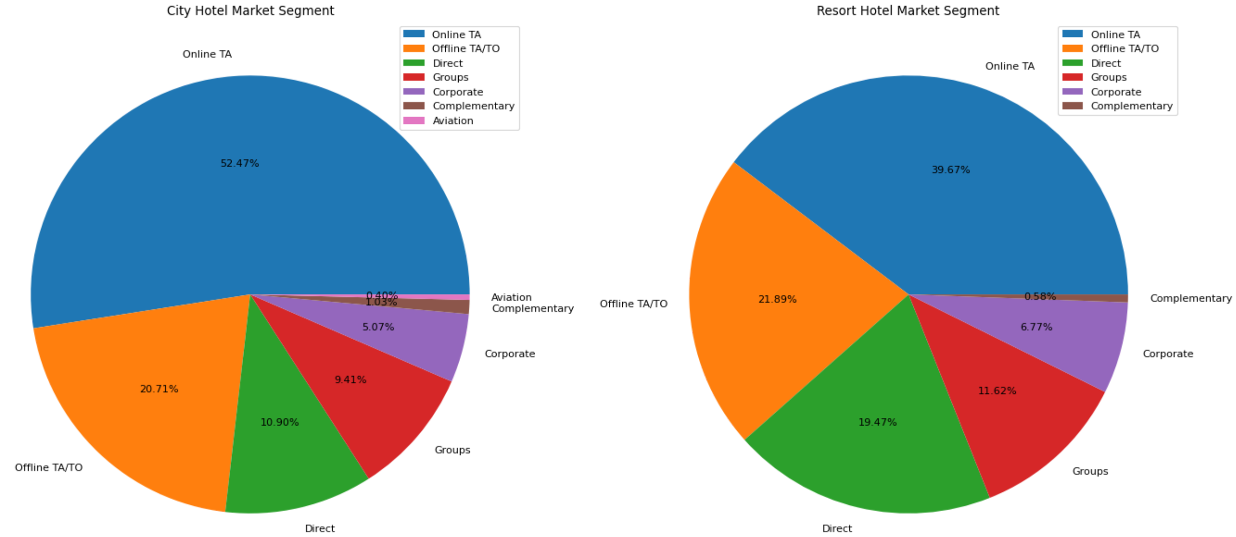


The main customer type of the hotel is transient travelers, accounting for about 70%.

1. Hotel booking method

In [13]:

```
#create pie charts to visualize the distribution of market segments for both City Hotel
city_segment=city_hotel.market_segment.value_counts()
resort_segment=resort_hotel.market_segment.value_counts()
plt.figure(figsize=(21,12),dpi=80)
plt.subplot(1,2,1)
plt.pie(city_segment,labels=city_segment.index,autopct='%.2f%%')
plt.legend()
plt.title('City Hotel Market Segment')
plt.subplot(1,2,2)
plt.pie(resort_segment,labels=resort_segment.index,autopct='%.2f%%')
plt.title('Resort Hotel Market Segment')
plt.legend()
plt.show()
```

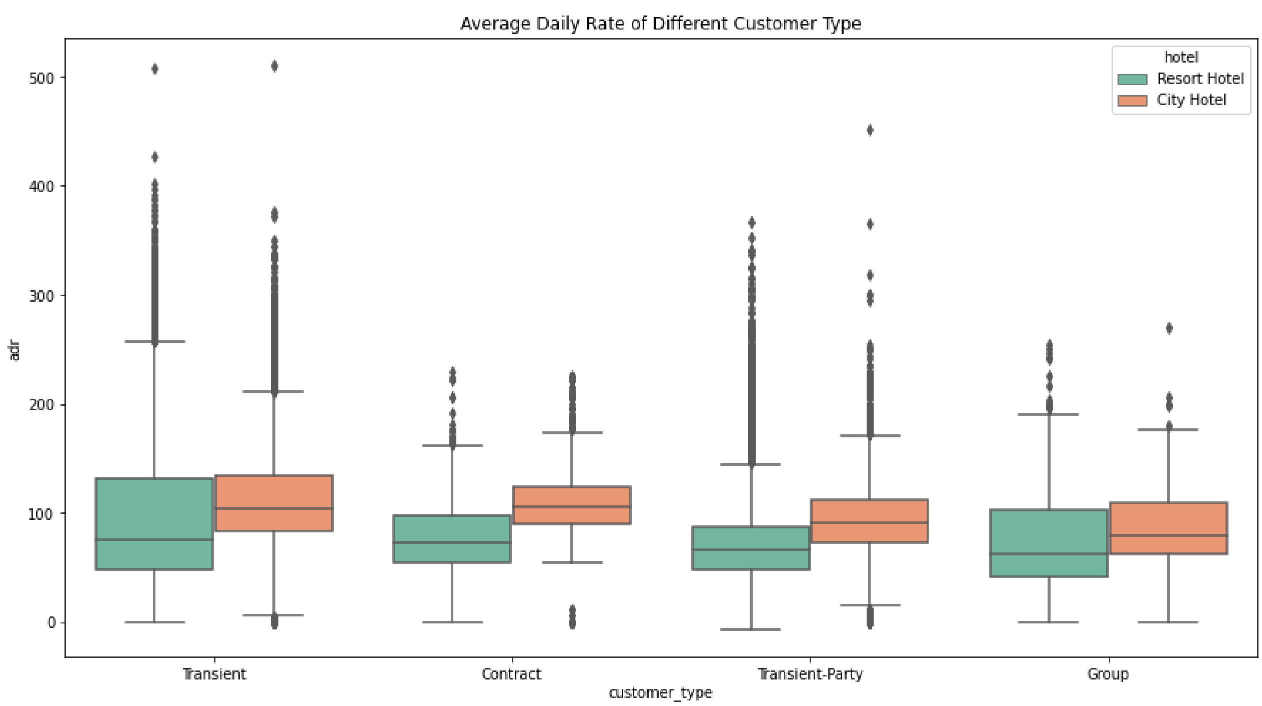


The customers of the two hotels mainly come from online travel agencies, which account for even more than 50% of the City Hotel; offline travel agencies come next, accounting for about 20%.

1. Average daily expenses of various types of passengers

In [14]:
```
#visualize the distribution of Average Daily Rate (adr) across different customer types,
plt.figure(figsize=(15,8))
sns.boxplot(x='customer_type'
            ,y='adr'
            ,hue='hotel'
            ,data=df[df.is_canceled==0]
            ,palette=sns.color_palette('Set2',2)
           )
plt.title('Average Daily Rate of Different Customer Type')
```

Out[14]:
```
Text(0.5, 1.0, 'Average Daily Rate of Different Customer Type')
```

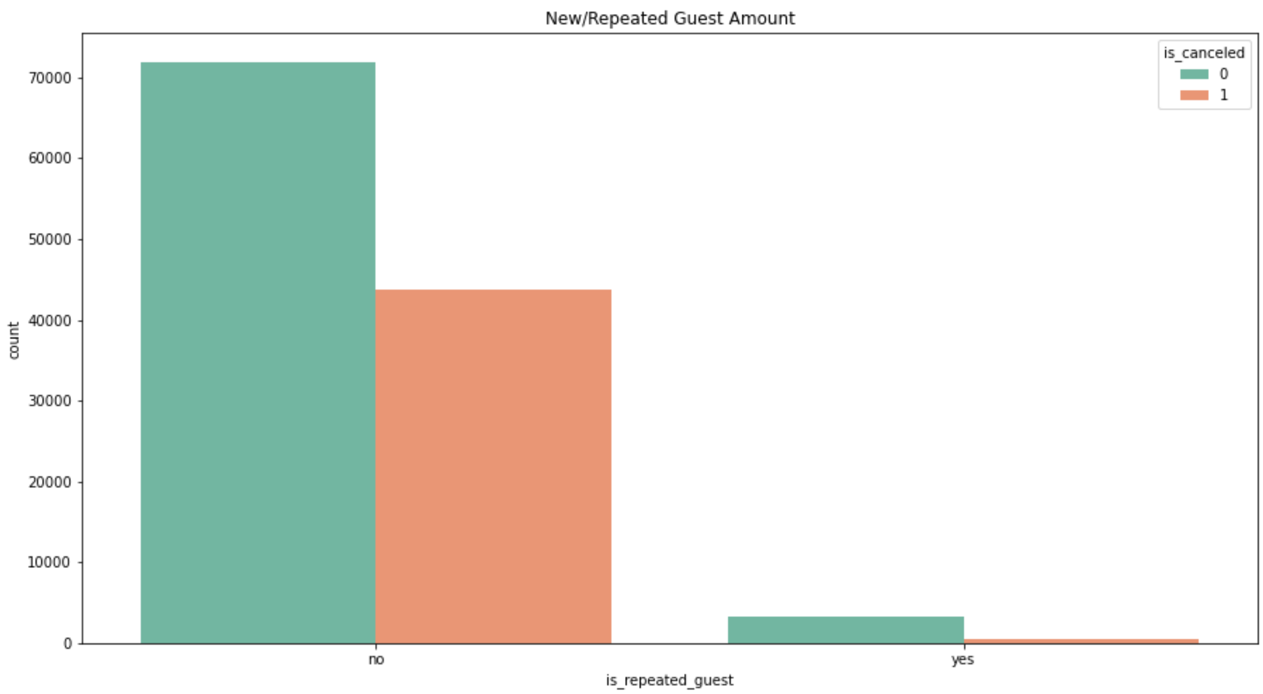


The average daily expenditure of all types of customers of City Hotel is higher than that of Resort Hotel; among the four types of customers, the consumption of individual travelers (Transient) is the highest and that of group travelers (Group) is the lowest.

7.Number of new and old customers and cancellation rate

In [15]:
```
# visualize the count of bookings, categorized by whether the guest is a repeated guest
plt.figure(figsize=(15,8))
sns.countplot(x='is_repeated_guest'
              ,data=df
              ,hue='is_canceled'
              ,palette=sns.color_palette('Set2',2)
             )
plt.title('New/Repeated Guest Amount')
plt.xticks(range(2),['no','yes'])
```

Out[15]:
```
([<matplotlib.axis.XTick at 0x2080ad6baf0>,
  <matplotlib.axis.XTick at 0x2080ad6bac0>],
 [Text(0, 0, 'no'), Text(1, 0, 'yes')])
```



In [16]:
```
#calculate and printing the cancellation rates for new and repeated guests
guest_cancel=(df.loc[df['is_canceled']==1]['is_repeated_guest'].value_counts()/df['is_r
guest_cancel.index=['New Guest', 'Repeated Guest']
print('Cancellation rate for new and old customers'.center(15),guest_cancel,sep='\n')
```

```
Cancellation rate for new and old customers
New Guest         0.377851
Repeated Guest    0.144882
Name: is_repeated_guest, dtype: float64
```

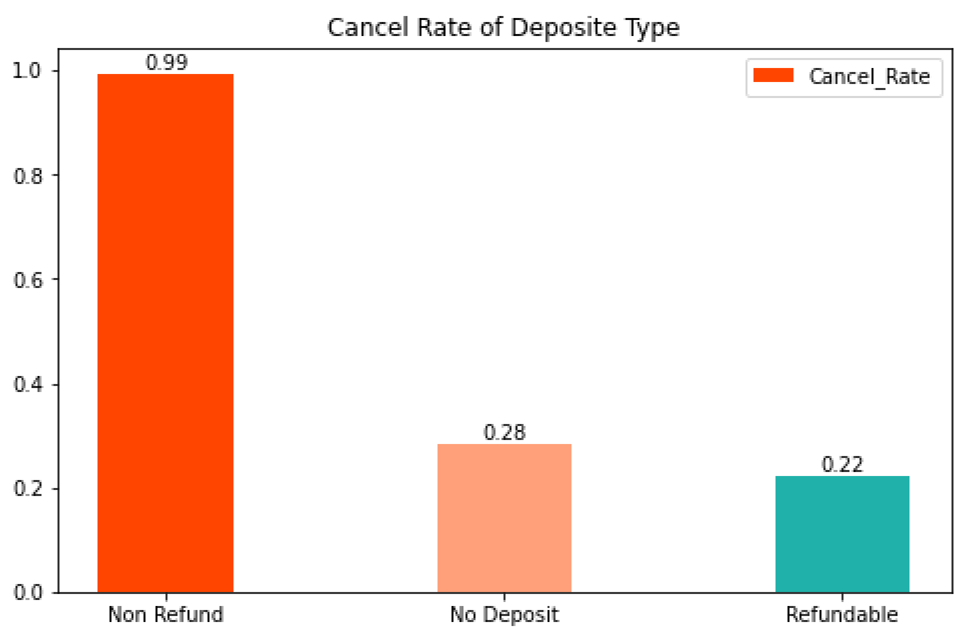
The cancellation rate for regular customers was 14.4%, while the cancellation rate for new customers reached 37.8%, which was 24 percentage points higher than that for regular customers.

1. Deposit method and reservation cancellation rate

In [17]:
```
print('Three deposit methods for booking quantity'.center(15),df['deposit_type'].value_
```

```
Three deposit methods for booking quantity
No Deposit    104641
Non Refund     14587
Refundable       162
Name: deposit_type, dtype: int64
```

In [18]:
```
#calculate the cancellation rates based on the 'deposit_type', and visualizing these ra
deposit_cancel=(df.loc[df['is_canceled']==1]['deposit_type'].value_counts()/df['deposit_
plt.figure(figsize=(8,5))
x=range(len(deposit_cancel.index))
y=deposit_cancel.values
plt.bar(x,y,label='Cancel_Rate',color=['orangered','lightsalmon','lightseagreen'],width
plt.xticks(x,deposit_cancel.index)
plt.legend()
plt.title('Cancel Rate of Deposite Type')
for x,y in zip(x,y):
    plt.text(x,y,'%.2f' % y,ha = 'center',va = 'bottom')
```

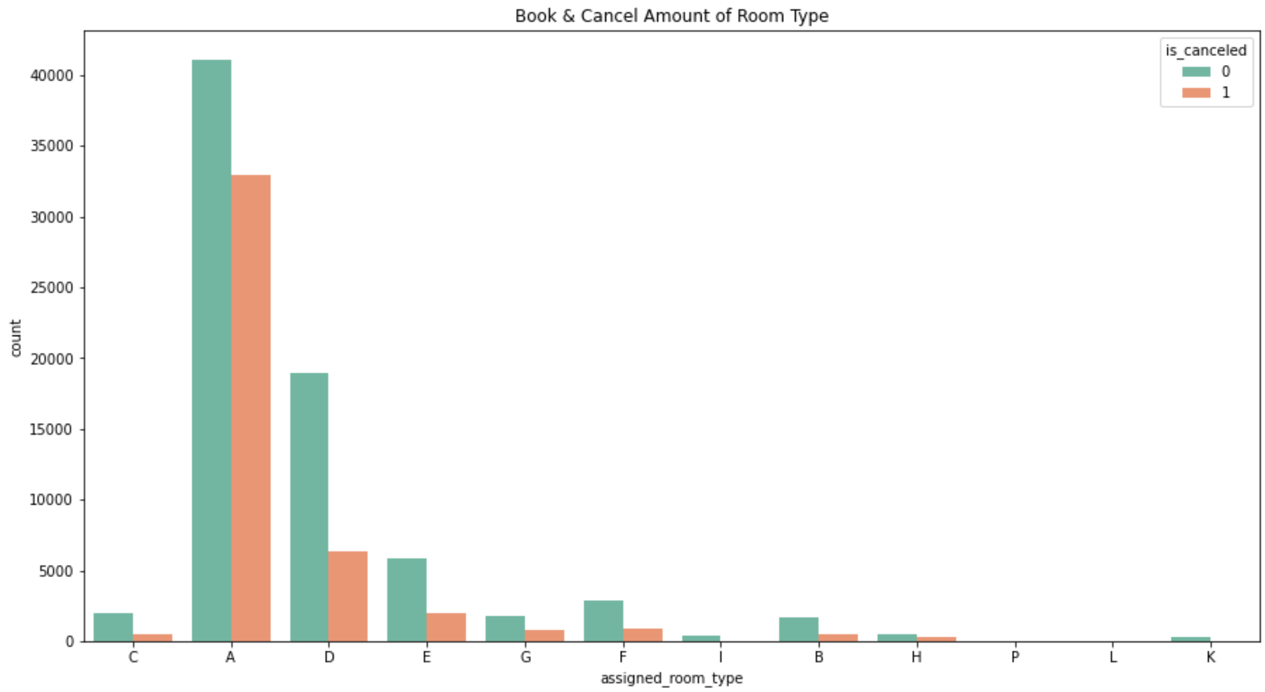


'No Deposit' is the method with the highest number of bookings and has a low cancellation rate, while the cancellation rate of non-refundable type is as high as 99%. This type of deposit method can be reduced to reduce Customer cancellation rate.

1. Room type and cancellation volume

In [19]:
```
#visualize the counts of bookings and cancellations across different assigned room type:
plt.figure(figsize=(15,8))
sns.countplot(x='assigned_room_type'
              ,data=df
              ,hue='is_canceled'
              ,palette=sns.color_palette('Set2',2)
             )
plt.title('Book & Cancel Amount of Room Type')
```

Out[19]:
```
Text(0.5, 1.0, 'Book & Cancel Amount of Room Type')
```



In [20]:
```
#calculate cancellation rates for the top 7 assigned room types and printing them in de
room_cancel=df.loc[df['is_canceled']==1]['assigned_room_type'].value_counts()[:7]/df['a
print('Cancellation rates for different room types'.center(5),room_cancel.sort_values(a
```

```
Cancellation rates for different room types
A    0.444925
G    0.305523
E    0.252114
D    0.251244
F    0.247134
B    0.236708
C    0.187789
Name: assigned_room_type, dtype: float64
```

Among the top seven room types with the most bookings, the cancellation rates of room types A and G are higher than other room types, and the cancellation rate of room type A is as high as 44.5%.

Conclusion

1. The booking volume and cancellation rate of City Hotel are much higher than that of Resort Hotel. The hotel should conduct customer surveys to gain an in-depth understanding of the factors that cause customers to give up on bookings in order to reduce customer cancellation rates.

2. Hotels should make good use of the peak tourist season of July and August every year. They can increase prices appropriately while ensuring service quality to obtain more profits, and conduct preferential activities during the off-season (winter), such as Christmas sales and New Year activities, to reduce Hotel vacancy rate.

3. Hotels need to analyze customer profiles from major source countries such as Portugal and the United Kingdom, understand the attribute tags, preferences and consumption characteristics of these customers, and launch exclusive services to reduce customer cancellation rates.

4. Since individual travelers are the main customer group of hotels and have high consumption levels, hotels can increase the promotion and marketing of independent travelers through online and offline travel agencies, thereby attracting more tourists of this type.

5. The cancellation rate of new customers is 24% higher than that of old customers. Therefore, hotels should focus on the booking and check-in experience of new customers, and provide more guidance and benefits to new customers, such as providing discounts to first-time customers and conducting research on new customers. Provide feedback on satisfaction and dissatisfaction with your stay to improve future services and maintain good old customers.

6. The cancellation rate of non-refundable deposits is as high as 99%. Hotels should optimize this method, such as returning 50% of the deposit, or cancel this method directly to increase the occupancy rate.

7. The cancellation rate of room types A and G is much higher than that of other room types. The hotel should carefully confirm the room information with the customer when making a reservation, so that the customer can fully understand the room situation, avoid cognitive errors, and at the same time be able to understand the room facilities. Optimize and improve service levels.

Data Processing (Week 4)

In [21]:

```
#create a new DataFrame 'df1' from 'df'
df1=df.drop(labels=['reservation_status_date'],axis=1)
```

Handling Categorical Variables

In [22]:

```
cate=df1.columns[df1.dtypes == "object"].tolist() #getting the names of all columns in
#categorical variables expressed as numbers
num_cate=['agent','company','is_repeated_guest']
cate=cate+num_cate
```

In [23]:

```
import numpy as np #linear algebra
#creating a dictionary
results={}
for i in ['agent','company']:
    result=np.sort(df1[i].unique())
    results[i]=result
results
```

Out[23]:

```
{'agent': array([  1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.,  11.,
         12.,  13.,  14.,  15.,  16.,  17.,  19.,  20.,  21.,  22.,  23.,
         24.,  25.,  26.,  27.,  28.,  29.,  30.,  31.,  32.,  33.,  34.,
         35.,  36.,  37.,  38.,  39.,  40.,  41.,  42.,  44.,  45.,  47.,
```

```
        50.,  52.,  53.,  54.,  55.,  56.,  57.,  58.,  59.,  60.,  61.,
        63.,  64.,  66.,  67.,  68.,  69.,  70.,  71.,  72.,  73.,  74.,
        75.,  77.,  78.,  79.,  81.,  82.,  83.,  85.,  86.,  87.,  88.,
        89.,  90.,  91.,  92.,  93.,  94.,  95.,  96.,  98.,  99., 103.,
       104., 105., 106., 107., 110., 111., 112., 114., 115., 117., 118.,
       119., 121., 122., 126., 127., 128., 129., 132., 133., 134., 135.,
       138., 139., 141., 142., 143., 144., 146., 147., 148., 149., 150.,
       151., 152., 153., 154., 155., 156., 157., 158., 159., 162., 163.,
       165., 167., 168., 170., 171., 173., 174., 175., 177., 179., 180.,
       181., 182., 183., 184., 185., 187., 191., 192., 193., 195., 196.,
       197., 201., 205., 208., 210., 211., 213., 214., 215., 216., 219.,
       220., 223., 227., 229., 232., 234., 235., 236., 240., 241., 242.,
       243., 244., 245., 247., 248., 249., 250., 251., 252., 253., 254.,
       256., 257., 258., 261., 262., 265., 267., 269., 270., 273., 275.,
       276., 278., 280., 281., 282., 283., 285., 286., 287., 288., 289.,
       290., 291., 294., 295., 296., 298., 299., 300., 301., 302., 303.,
       304., 305., 306., 307., 308., 310., 313., 314., 315., 321., 323.,
       324., 325., 326., 327., 328., 330., 331., 332., 333., 334., 335.,
       336., 337., 339., 341., 344., 346., 348., 350., 352., 354., 355.,
       358., 359., 360., 363., 364., 367., 368., 370., 371., 375., 378.,
       384., 385., 387., 388., 390., 391., 393., 394., 397., 403., 404.,
       405., 406., 408., 410., 411., 414., 416., 418., 420., 423., 425.,
       426., 427., 429., 430., 431., 432., 433., 434., 436., 438., 440.,
       441., 444., 446., 449., 450., 451., 453., 454., 455., 459., 461.,
       464., 467., 468., 469., 472., 474., 475., 476., 479., 480., 481.,
       483., 484., 492., 493., 495., 497., 502., 508., 509., 510., 526.,
       527., 531., 535.,  nan]),
'company': array([  6.,   8.,   9.,  10.,  11.,  12.,  14.,  16.,  18.,  20.,  22.,
        28.,  29.,  31.,  32.,  34.,  35.,  37.,  38.,  39.,  40.,  42.,
        43.,  45.,  46.,  47.,  48.,  49.,  51.,  52.,  53.,  54.,  59.,
        61.,  62.,  64.,  65.,  67.,  68.,  71.,  72.,  73.,  76.,  77.,
        78.,  80.,  81.,  82.,  83.,  84.,  85.,  86.,  88.,  91.,  92.,
        93.,  94.,  96.,  99., 100., 101., 102., 103., 104., 105., 106.,
       107., 108., 109., 110., 112., 113., 115., 116., 118., 120., 122.,
       126., 127., 130., 132., 135., 137., 139., 140., 142., 143., 144.,
       146., 148., 149., 150., 153., 154., 158., 159., 160., 163., 165.,
       167., 168., 169., 174., 178., 179., 180., 183., 184., 185., 186.,
       192., 193., 195., 197., 200., 202., 203., 204., 207., 209., 210.,
       212., 213., 215., 216., 217., 218., 219., 220., 221., 222., 223.,
       224., 225., 227., 229., 230., 232., 233., 234., 237., 238., 240.,
       242., 243., 245., 246., 250., 251., 253., 254., 255., 257., 258.,
       259., 260., 263., 264., 268., 269., 270., 271., 272., 273., 274.,
       275., 277., 278., 279., 280., 281., 282., 284., 286., 287., 288.,
       289., 290., 291., 292., 293., 297., 301., 302., 304., 305., 307.,
       308., 309., 311., 312., 313., 314., 316., 317., 318., 319., 320.,
       321., 323., 324., 325., 329., 330., 331., 332., 333., 334., 337.,
       338., 341., 342., 343., 346., 347., 348., 349., 350., 351., 352.,
       353., 355., 356., 357., 358., 360., 361., 362., 364., 365., 366.,
       367., 368., 369., 370., 371., 372., 373., 376., 377., 378., 379.,
       380., 382., 383., 384., 385., 386., 388., 390., 391., 392., 393.,
       394., 395., 396., 397., 398., 399., 400., 401., 402., 403., 405.,
       407., 408., 409., 410., 411., 412., 413., 415., 416., 417., 418.,
       419., 420., 421., 422., 423., 424., 425., 426., 428., 429., 433.,
       435., 436., 437., 439., 442., 443., 444., 445., 446., 447., 448.,
       450., 451., 452., 454., 455., 456., 457., 458., 459., 460., 461.,
       465., 466., 470., 477., 478., 479., 481., 482., 483., 484., 485.,
       486., 487., 489., 490., 491., 492., 494., 496., 497., 498., 499.,
       501., 504., 506., 507., 511., 512., 513., 514., 515., 516., 518.,
       520., 521., 523., 525., 528., 530., 531., 534., 539., 541., 543.,
        nan])}
```

In [24]:
```
# the agent and company columns have a large number of empty values and no 0 values, so
df1[['agent','company']]=df1[['agent','company']].fillna(0,axis=0)
```

In [25]:
```
df1.loc[:,cate].isnull().mean()
```

Out[25]:
```
hotel                   0.000000
arrival_date_month      0.000000
meal                    0.000000
country                 0.004087
market_segment          0.000000
distribution_channel    0.000000
reserved_room_type      0.000000
assigned_room_type      0.000000
deposit_type            0.000000
customer_type           0.000000
reservation_status      0.000000
name                    0.000000
email                   0.000000
phone-number            0.000000
credit_card             0.000000
agent                   0.000000
company                 0.000000
is_repeated_guest       0.000000
dtype: float64
```

In [26]:
```
#create new variables in_company and in_agent to classify passengers. If company and age
df1.loc[df1['company'] == 0,'in_company']='NO'
df1.loc[df1['company'] != 0,'in_company']='YES'
df1.loc[df1['agent'] == 0,'in_agent']='NO'
df1.loc[df1['agent'] != 0,'in_agent']='YES'
```

In [27]:
```
#create a new feature same_assignment. If the booked room type is consistent with the a
df1.loc[df1['reserved_room_type'] == df1['assigned_room_type'],'same_assignment']='Yes'
df1.loc[df1['reserved_room_type'] != df1['assigned_room_type'],'same_assignment']='No'
```

In [28]:
```
#delete four features except 'reserved_room_type', 'assigned_room_type', 'agent', 'compa
df1=df1.drop(labels=['reserved_room_type','assigned_room_type','agent','company'],axis=
```

In [29]:
```
#reset 'is_repeated_guest', frequent guests are marked as YES, non-repeated guests are n
df1['is_repeated_guest'][df1['is_repeated_guest']==0]='NO'
df1['is_repeated_guest'][df1['is_repeated_guest']==1]='YES'
```

In [30]:
```
#filling the missing values in the 'country' column of the DataFrame 'df1' with the mode
df1['country']=df1['country'].fillna(df1['country'].mode()[0])
```

In [31]:
```
for i in ['in_company','in_agent','same_assignment']:
    cate.append(i)

for i in ['reserved_room_type','assigned_room_type','agent','company']:
```

```
    cate.remove(i)
cate
```

Out[31]:

```
['hotel',
 'arrival_date_month',
 'meal',
 'country',
 'market_segment',
 'distribution_channel',
 'deposit_type',
 'customer_type',
 'reservation_status',
 'name',
 'email',
 'phone-number',
 'credit_card',
 'is_repeated_guest',
 'in_company',
 'in_agent',
 'same_assignment']
```

In [32]:

```
#encoding categorical features
from sklearn.preprocessing import OrdinalEncoder
oe = OrdinalEncoder()
oe = oe.fit(df1.loc[:,cate])
df1.loc[:,cate] = oe.transform(df1.loc[:,cate])
```

Working With Continuous Variables

In [33]:

```
#to filter out continuous variables, you need to delete the label 'is_canceled' first.
col=df1.columns.tolist()
col.remove('is_canceled')
for i in cate:
    col.remove(i)
col
```

Out[33]:

```
['lead_time',
 'arrival_date_year',
 'arrival_date_week_number',
 'arrival_date_day_of_month',
 'stays_in_weekend_nights',
 'stays_in_week_nights',
 'adults',
 'children',
 'babies',
 'previous_cancellations',
 'previous_bookings_not_canceled',
 'booking_changes',
 'days_in_waiting_list',
 'adr',
 'required_car_parking_spaces',
 'total_of_special_requests']
```

In [34]:

```
df1[col].isnull().sum()
```

Out[34]:

```
lead_time                    0
arrival_date_year            0
```

```
arrival_date_week_number          0
arrival_date_day_of_month         0
stays_in_weekend_nights           0
stays_in_week_nights              0
adults                            0
children                          4
babies                            0
previous_cancellations            0
previous_bookings_not_canceled    0
booking_changes                   0
days_in_waiting_list              0
adr                               0
required_car_parking_spaces       0
total_of_special_requests         0
dtype: int64
```

In [35]:

```
#use mode to fill null values in xtrain children column
df1['children']=df1['children'].fillna(df1['children'].mode()[0])
```

In [36]:

```
#continuous variables are dimensionless
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
ss = ss.fit(df1.loc[:,col])
df1.loc[:,col] = ss.transform(df1.loc[:,col])
```

Correlation Coefficient of Each Variable

In [37]:

```
#calculating the correlation of all numerical columns with the 'is_canceled column' in 
cor=df1.corr()
cor=abs(cor['is_canceled']).sort_values()
cor
```

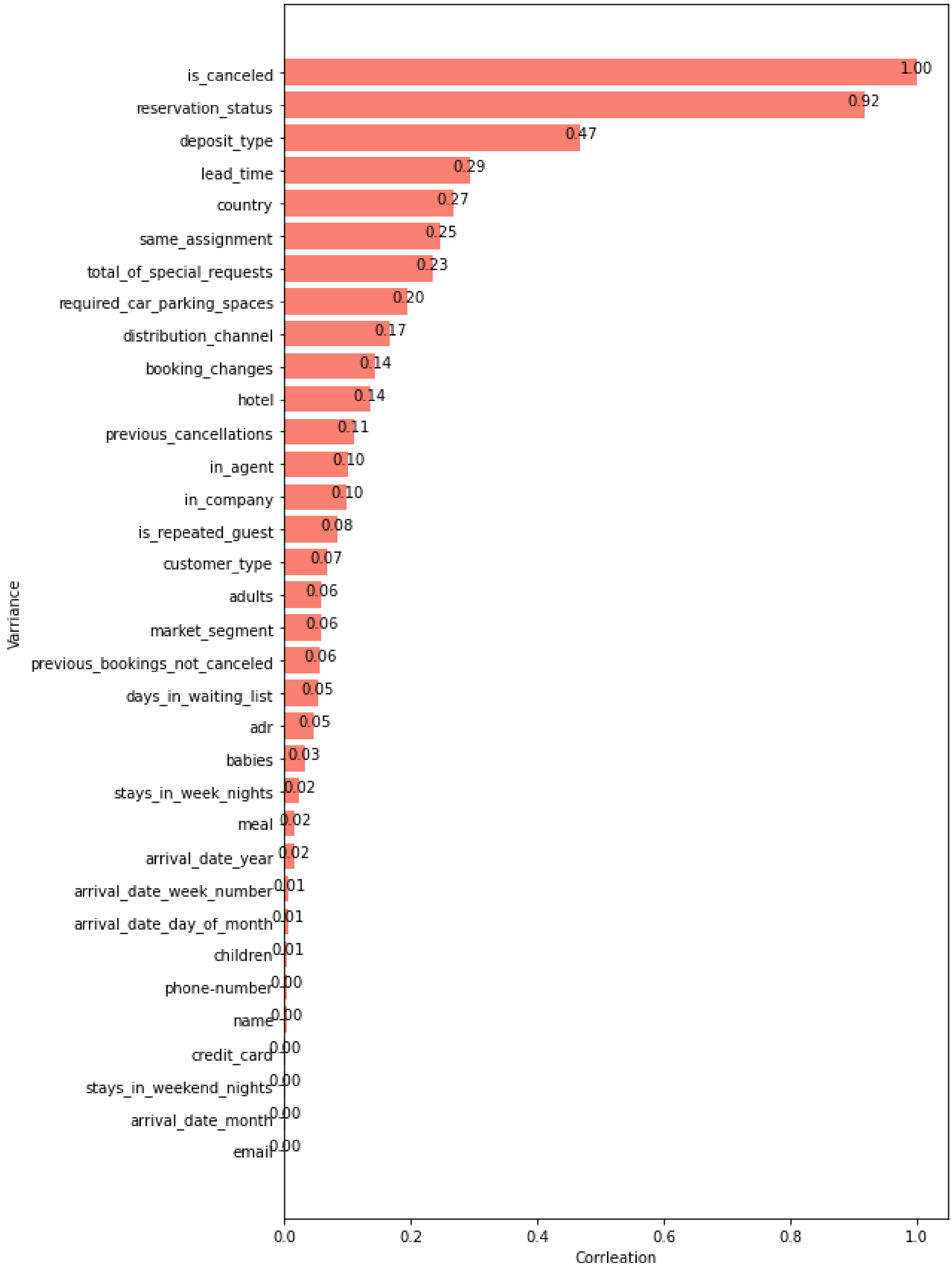
Out[37]:

```
email                             0.000723
arrival_date_month                0.001491
stays_in_weekend_nights           0.001791
credit_card                       0.002515
name                              0.004253
phone-number                      0.004342
children                          0.005036
arrival_date_day_of_month         0.006130
arrival_date_week_number          0.008148
arrival_date_year                 0.016660
meal                              0.017678
stays_in_week_nights              0.024765
babies                            0.032491
adr                               0.047557
days_in_waiting_list              0.054186
previous_bookings_not_canceled    0.057358
market_segment                    0.059338
adults                            0.060017
customer_type                     0.068140
is_repeated_guest                 0.084793
in_company                        0.099310
in_agent                          0.102068
previous_cancellations            0.110133
hotel                             0.136531
```

```
booking_changes                0.144381
distribution_channel           0.167600
required_car_parking_spaces    0.195498
total_of_special_requests      0.234658
same_assignment                0.247770
country                        0.267502
lead_time                      0.293123
deposit_type                   0.468634
reservation_status             0.917196
is_canceled                    1.000000
Name: is_canceled, dtype: float64
```

In [38]:

```
#create a horizontal bar plot using Matplotlib to visualize the absolute correlation va
plt.figure(figsize=(8,15))
x=range(len(cor.index))
name=cor.index
y=abs(cor.values)
plt.barh(x,y,color='salmon')
plt.yticks(x,name)
for x,y in zip(x,y):
    plt.text(y,x-0.1,'%.2f' % y,ha = 'center',va = 'bottom')
plt.xlabel('Corrleation')
plt.ylabel('Varriance')
plt.show()
```



The reservation status ('reservation_status') has the highest correlation with whether to cancel the reservation, reaching 0.92, but considering that it may cause the model to overfit in the future, it is deleted; the deposit type ('deposit_type') reaches 0.47, creating a characteristic Whether the reservation and assigned room type are consistent ('same_assignment') also has a correlation of 0.25.

In [39]:
```
#copy 'df1' with the column labeled 'reservation_status' dropped.
df2=df1.drop('reservation_status',axis=1)
```

Week 5

In [40]:
```
#dropping columns that are not useful
useless_col = ['email', 'phone-number', 'credit_card', 'name', 'days_in_waiting_list',
               'reservation_status', 'country', 'days_in_waiting_list']

df.drop(useless_col, axis = 1, inplace = True)
```

In [41]:
```
df.head()
```

Out[41]:

	hotel	is_canceled	lead_time	arrival_date_month	arrival_date_week_number	arrival_date_day_of_mont
0	Resort Hotel	0	342	July	27	
1	Resort Hotel	0	737	July	27	
2	Resort Hotel	0	7	July	27	
3	Resort Hotel	0	13	July	27	
4	Resort Hotel	0	14	July	27	

5 rows × 26 columns

In [42]:
```
# creating numerical and categorical dataframes
cat_cols = [col for col in df.columns if df[col].dtype == 'O']
cat_cols
```

Out[42]:
```
['hotel',
 'arrival_date_month',
 'meal',
 'market_segment',
 'distribution_channel',
 'reserved_room_type',
 'deposit_type',
 'customer_type',
 'reservation_status_date']
```

In [43]:
```
cat_df = df[cat_cols]
cat_df.head()
```

Out[43]:

	hotel	arrival_date_month	meal	market_segment	distribution_channel	reserved_room_type	deposit_
0	Resort Hotel	July	BB	Direct	Direct	C	No De
1	Resort Hotel	July	BB	Direct	Direct	C	No De
2	Resort Hotel	July	BB	Direct	Direct	A	No De
3	Resort Hotel	July	BB	Corporate	Corporate	A	No De
4	Resort Hotel	July	BB	Online TA	TA/TO	A	No De

In [44]:

```
#Convert 'reservation_status_date' to DateTime type
cat_df['reservation_status_date'] = pd.to_datetime(cat_df['reservation_status_date'])
#Extract the Year from the 'reservation_status_date'
cat_df['year'] = cat_df['reservation_status_date'].dt.year
#Extract the Month from the 'reservation_status_date'
cat_df['month'] = cat_df['reservation_status_date'].dt.month
#Extract the Day from the 'reservation_status_date'
cat_df['day'] = cat_df['reservation_status_date'].dt.day
```

In [45]:

```
cat_df.drop(['reservation_status_date','arrival_date_month'] , axis = 1, inplace = True
```

In [46]:

```
cat_df.head(15)
```

Out[46]:

	hotel	meal	market_segment	distribution_channel	reserved_room_type	deposit_type	customer_typ
0	Resort Hotel	BB	Direct	Direct	C	No Deposit	Transier
1	Resort Hotel	BB	Direct	Direct	C	No Deposit	Transier
2	Resort Hotel	BB	Direct	Direct	A	No Deposit	Transier
3	Resort Hotel	BB	Corporate	Corporate	A	No Deposit	Transier
4	Resort Hotel	BB	Online TA	TA/TO	A	No Deposit	Transier
5	Resort Hotel	BB	Online TA	TA/TO	A	No Deposit	Transier
6	Resort Hotel	BB	Direct	Direct	C	No Deposit	Transier
7	Resort Hotel	FB	Direct	Direct	C	No Deposit	Transier

	hotel	meal	market_segment	distribution_channel	reserved_room_type	deposit_type	customer_typ
8	Resort Hotel	BB	Online TA	TA/TO	A	No Deposit	Transier
9	Resort Hotel	HB	Offline TA/TO	TA/TO	D	No Deposit	Transier
10	Resort Hotel	BB	Online TA	TA/TO	E	No Deposit	Transier
11	Resort Hotel	HB	Online TA	TA/TO	D	No Deposit	Transier
12	Resort Hotel	BB	Online TA	TA/TO	D	No Deposit	Transier
13	Resort Hotel	HB	Online TA	TA/TO	G	No Deposit	Transier
14	Resort Hotel	BB	Online TA	TA/TO	E	No Deposit	Transier

In [47]:

```
# printing unique values of each column
for col in cat_df.columns:
    print(f"{col}: \n{cat_df[col].unique()}\n")
```

```
hotel:
['Resort Hotel' 'City Hotel']

meal:
['BB' 'FB' 'HB' 'SC' 'Undefined']

market_segment:
['Direct' 'Corporate' 'Online TA' 'Offline TA/TO' 'Complementary' 'Groups'
 'Undefined' 'Aviation']

distribution_channel:
['Direct' 'Corporate' 'TA/TO' 'Undefined' 'GDS']

reserved_room_type:
['C' 'A' 'D' 'E' 'G' 'F' 'H' 'L' 'P' 'B']

deposit_type:
['No Deposit' 'Refundable' 'Non Refund']

customer_type:
['Transient' 'Contract' 'Transient-Party' 'Group']

year:
[2015 2014 2016 2017]

month:
[ 7  5  4  6  3  8  9  1 11 10 12  2]

day:
[ 1  2  3  6 22 23  5  7  8 11 15 16 29 19 18  9 13  4 12 26 17 10 20 14
 30 28 25 21 27 24 31]
```

In [48]:

```python
# encoding categorical variables, which can be in text/string format, into numerical fo

cat_df['hotel'] = cat_df['hotel'].map({'Resort Hotel' : 0, 'City Hotel' : 1})

cat_df['meal'] = cat_df['meal'].map({'BB' : 0, 'FB': 1, 'HB': 2, 'SC': 3, 'Undefined':

cat_df['market_segment'] = cat_df['market_segment'].map({'Direct': 0, 'Corporate': 1, '
                                                            'Complementary': 4, 'Groups'

cat_df['distribution_channel'] = cat_df['distribution_channel'].map({'Direct': 0, 'Corp
                                                                        'GDS': 4})

cat_df['reserved_room_type'] = cat_df['reserved_room_type'].map({'C': 0, 'A': 1, 'D': 2
                                                                    'L': 7, 'B': 8})

cat_df['deposit_type'] = cat_df['deposit_type'].map({'No Deposit': 0, 'Refundable': 1,

cat_df['customer_type'] = cat_df['customer_type'].map({'Transient': 0, 'Contract': 1, '

cat_df['year'] = cat_df['year'].map({2015: 0, 2014: 1, 2016: 2, 2017: 3})
```

In [49]:

```python
cat_df.head(15)
```

Out[49]:

	hotel	meal	market_segment	distribution_channel	reserved_room_type	deposit_type	customer_type
0	0	0	0	0	0.0	0	(
1	0	0	0	0	0.0	0	(
2	0	0	0	0	1.0	0	(
3	0	0	1	1	1.0	0	(
4	0	0	2	2	1.0	0	(
5	0	0	2	2	1.0	0	(
6	0	0	0	0	0.0	0	(
7	0	1	0	0	0.0	0	(
8	0	0	2	2	1.0	0	(
9	0	2	3	2	2.0	0	(
10	0	0	2	2	3.0	0	(
11	0	2	2	2	2.0	0	(
12	0	0	2	2	2.0	0	(
13	0	2	2	2	4.0	0	(
14	0	0	2	2	3.0	0	(

In [50]:

```
num_df = df.drop(columns = cat_cols, axis = 1)
num_df.drop('is_canceled', axis = 1, inplace = True)
num_df
```

Out[50]:

	lead_time	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_i
0	342	27	1	0	
1	737	27	1	0	
2	7	27	1	0	
3	13	27	1	0	
4	14	27	1	0	
...	...	...	...	...	
119385	23	35	30	2	
119386	102	35	31	2	
119387	34	35	31	2	
119388	109	35	31	2	
119389	205	35	29	2	

119390 rows × 16 columns

In [51]:
```
num_df.var()
```

Out[51]:
```
lead_time                         11419.721511
arrival_date_week_number            185.099790
arrival_date_day_of_month            77.102966
stays_in_weekend_nights               0.997229
stays_in_week_nights                  3.641554
adults                                0.335543
children                              0.158851
babies                                0.009494
is_repeated_guest                     0.030894
previous_cancellations                0.712904
previous_bookings_not_canceled        2.242317
agent                             12271.000405
company                           17333.042879
adr                                2553.866100
required_car_parking_spaces           0.060168
total_of_special_requests             0.628529
dtype: float64
```

In [52]:
```
# normalizing numerical variables, uses the natural logarithm to transform the data.
#It's essential to add 1 before taking the log to handle instances where the column valu
num_df['lead_time'] = np.log(num_df['lead_time'] + 1)
num_df['arrival_date_week_number'] = np.log(num_df['arrival_date_week_number'] + 1)
num_df['arrival_date_day_of_month'] = np.log(num_df['arrival_date_day_of_month'] + 1)
num_df['agent'] = np.log(num_df['agent'] + 1)
num_df['company'] = np.log(num_df['company'] + 1)
num_df['adr'] = np.log(num_df['adr'] + 1)
```

In [53]:
```
num_df.var()
```

Out[53]:
```
lead_time                         2.591420
arrival_date_week_number          0.441039
arrival_date_day_of_month         0.506267
stays_in_weekend_nights           0.997229
stays_in_week_nights              3.641554
adults                            0.335543
children                          0.158851
babies                            0.009494
is_repeated_guest                 0.030894
previous_cancellations            0.712904
previous_bookings_not_canceled    2.242317
agent                             2.536204
company                           0.755665
adr                               0.540353
required_car_parking_spaces       0.060168
total_of_special_requests         0.628529
dtype: float64
```

In [54]:
```
num_df['adr'] = num_df['adr'].fillna(value = num_df['adr'].mean())
num_df.head(15)
```

Out[54]:

	lead_time	arrival_date_week_number	arrival_date_day_of_month	stays_in_weekend_nights	stays_in_we
0	5.837730	3.332205	0.693147	0	
1	6.603944	3.332205	0.693147	0	
2	2.079442	3.332205	0.693147	0	
3	2.639057	3.332205	0.693147	0	
4	2.708050	3.332205	0.693147	0	
5	2.708050	3.332205	0.693147	0	
6	0.000000	3.332205	0.693147	0	
7	2.302585	3.332205	0.693147	0	
8	4.454347	3.332205	0.693147	0	
9	4.330733	3.332205	0.693147	0	
10	3.178054	3.332205	0.693147	0	
11	3.583519	3.332205	0.693147	0	
12	4.234107	3.332205	0.693147	0	
13	2.944439	3.332205	0.693147	0	
14	3.637586	3.332205	0.693147	0	

Prepare the independent and dependent variables for a modeling task

In [55]:

```
#merging categorical and numerical dataframes
#X = pd.concat([cat_df, num_df], axis = 1)
#y = df['is_canceled']
x=df2.loc[:,df2.columns != 'is_canceled' ]
y=df2.loc[:,'is_canceled']
from sklearn.model_selection import train_test_split as tts
xtrain,xtest,ytrain,ytest=tts(x,y,test_size=0.3,random_state=90)
for i in [xtrain,xtest,ytrain,ytest]:
    i.index=range(i.shape[0])
```

In [56]:

```
x.shape, y.shape
```

Out[56]:

```
((119390, 32), (119390,))
```

In [58]:

```
xtrain.head()
```

Out[58]:

	hotel	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_
0	0.0	1.029252	1.192195	5.0	0.061361	-0.5
1	0.0	0.102829	-0.221286	8.0	-0.600156	-1.5

	hotel	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_r
2	1.0	0.168334	1.192195	6.0	-0.085642	1.6
3	1.0	0.767233	1.192195	5.0	-0.012141	-0.8
4	0.0	-0.421208	-0.221286	11.0	0.943385	1.1

5 rows × 32 columns

In [59]:

```
xtest.head()
```

Out[59]:

	hotel	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number	arrival_date_day_of_r
0	0.0	-0.963961	-1.634768	2.0	1.898910	1.2
1	0.0	-0.861025	-0.221286	5.0	0.208365	0.3
2	0.0	1.431638	-0.221286	5.0	0.355369	1.7
3	0.0	-0.879741	-0.221286	11.0	0.722879	-1.2
4	0.0	-0.224694	-0.221286	11.0	0.943385	1.1

5 rows × 32 columns

In [60]:

```
ytrain.head(), ytest.head()
```

Out[60]:

```
(0    1
 1    0
 2    0
 3    1
 4    0
 Name: is_canceled, dtype: int64,
 0    0
 1    0
 2    0
 3    0
 4    0
 Name: is_canceled, dtype: int64)
```

Week 6 (2XGBoost)

In [61]:

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\zhumh\anaconda3\lib\site-packages (2.
0.0)
Requirement already satisfied: numpy in c:\users\zhumh\anaconda3\lib\site-packages (from
xgboost) (1.22.4)
Requirement already satisfied: scipy in c:\users\zhumh\anaconda3\lib\site-packages (from
```

```
xgboost) (1.7.1)
Note: you may need to restart the kernel to use updated packages.

[notice] A new release of pip is available: 23.0 -> 23.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [62]:

```
import xgboost as xgb
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import cross_val_score as cvs,KFold
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score as AUC
from sklearn.model_selection import cross_val_score
```

In [63]:

```
#Define the Model Variations
#base Model:This is the default parameters
model_1 = xgb.XGBClassifier(objective='binary:logistic', random_state=90)

#model 2: Adjust tree related hyperparameters
model_2 = xgb.XGBClassifier(objective='binary:logistic', max_depth=5, min_child_weight=

#model 3: Adjust boosting related hyperparameters
model_3 = xgb.XGBClassifier(objective='binary:logistic', learning_rate=0.01, n_estimato
```

In [64]:

```
#Train and Evaluate Each Model
models = [model_1, model_2, model_3]
model_names = ['Base Model', 'Tree Hyperparameters', 'Boosting Hyperparameters']
results = []

for i, model in enumerate(models):
    model.fit(xtrain, ytrain)

    # Predict
    train_pred = model.predict_proba(xtrain)[:,1]
    val_pred = model.predict_proba(xtest)[:,1]

    # Evaluate
    train_auc = roc_auc_score(ytrain, train_pred)
    val_auc = roc_auc_score(ytest, val_pred)

    results.append([model_names[i], train_auc, val_auc])

# Print results
print("Model Variation | Train AUC | Validation AUC")
print("-----------------------------------------------")
for result in results:
    print(f"{result[0]:<25} | {result[1]:.4f}   | {result[2]:.4f}")
```

```
Model Variation | Train AUC | Validation AUC
----------------------------------------------
Base Model                | 0.9653   | 0.9429
Tree Hyperparameters      | 0.9550   | 0.9409
Boosting Hyperparameters  | 0.9302   | 0.9268
```

In [65]:

```
#Hyperparameter Tuning Using Optuna
import optuna
```

```
def objective(trial):
    learning_rate = trial.suggest_float("learning_rate", 1e-5, 1e-1)
    n_estimators = trial.suggest_int("n_estimators", 50, 500)
    max_depth = trial.suggest_int("max_depth", 1, 15)
    min_child_weight = trial.suggest_int("min_child_weight", 1, 7)
    subsample = trial.suggest_float("subsample", 0.5, 1.0)
    colsample_bytree = trial.suggest_float("colsample_bytree", 0.5, 1.0)

    model = xgb.XGBClassifier(
        objective='binary:logistic',
        learning_rate=learning_rate,
        n_estimators=n_estimators,
        max_depth=max_depth,
        min_child_weight=min_child_weight,
        subsample=subsample,
        colsample_bytree=colsample_bytree,
        random_state=90
    )
    cv = 5
    return cross_val_score(model, xtrain, ytrain, n_jobs=-1, cv=cv).mean()

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)

best_params = study.best_params
print("Best parameters:", best_params)
```

```
[I 2023-10-28 02:43:49,582] A new study created in memory with name: no-name-af860a4c-8d
31-445c-9da9-c112d45fcd6d
[I 2023-10-28 02:44:10,587] Trial 0 finished with value: 0.8730331625458602 and paramete
rs: {'learning_rate': 0.09082392401868805, 'n_estimators': 390, 'max_depth': 15, 'min_ch
ild_weight': 2, 'subsample': 0.5884747394039884, 'colsample_bytree': 0.789814926090966
5}. Best is trial 0 with value: 0.8730331625458602.
[I 2023-10-28 02:44:21,638] Trial 1 finished with value: 0.8594043815951569 and paramete
rs: {'learning_rate': 0.018460752721006418, 'n_estimators': 487, 'max_depth': 6, 'min_ch
ild_weight': 4, 'subsample': 0.6575145907539541, 'colsample_bytree': 0.893815054909217
3}. Best is trial 0 with value: 0.8730331625458602.
[I 2023-10-28 02:44:29,981] Trial 2 finished with value: 0.8667392683749137 and paramete
rs: {'learning_rate': 0.08135629185099287, 'n_estimators': 478, 'max_depth': 5, 'min_chi
ld_weight': 5, 'subsample': 0.7282272489432003, 'colsample_bytree': 0.8757220607945977}.
Best is trial 0 with value: 0.8730331625458602.
[I 2023-10-28 02:44:42,685] Trial 3 finished with value: 0.8713221324307648 and paramete
rs: {'learning_rate': 0.01611808563622525, 'n_estimators': 276, 'max_depth': 13, 'min_ch
ild_weight': 5, 'subsample': 0.6705430033177545, 'colsample_bytree': 0.541952757099653
2}. Best is trial 0 with value: 0.8730331625458602.
[I 2023-10-28 02:44:47,418] Trial 4 finished with value: 0.8177043188371051 and paramete
rs: {'learning_rate': 0.05663960332463452, 'n_estimators': 346, 'max_depth': 1, 'min_chi
ld_weight': 7, 'subsample': 0.5931125330153034, 'colsample_bytree': 0.7416997785867809}.
Best is trial 0 with value: 0.8730331625458602.
[I 2023-10-28 02:44:55,444] Trial 5 finished with value: 0.8378065071147687 and paramete
rs: {'learning_rate': 0.008840846939531093, 'n_estimators': 432, 'max_depth': 4, 'min_ch
ild_weight': 6, 'subsample': 0.8624553634919301, 'colsample_bytree': 0.592363739092985
2}. Best is trial 0 with value: 0.8730331625458602.
[I 2023-10-28 02:45:02,755] Trial 6 finished with value: 0.8486951862573289 and paramete
rs: {'learning_rate': 0.07537822473569353, 'n_estimators': 471, 'max_depth': 2, 'min_chi
ld_weight': 1, 'subsample': 0.55936726245225, 'colsample_bytree': 0.5188098838476806}. B
est is trial 0 with value: 0.8730331625458602.
[I 2023-10-28 02:45:10,373] Trial 7 finished with value: 0.8725784675201756 and paramete
rs: {'learning_rate': 0.0625273156521673, 'n_estimators': 203, 'max_depth': 10, 'min_chi
ld_weight': 5, 'subsample': 0.7151118942711387, 'colsample_bytree': 0.9583260448874129}.
```

```
Best is trial 0 with value: 0.8730331625458602.
[I 2023-10-28 02:45:13,511] Trial 8 finished with value: 0.8418389293998224 and paramete
rs: {'learning_rate': 0.09944110088443948, 'n_estimators': 175, 'max_depth': 2, 'min_chi
ld_weight': 4, 'subsample': 0.8260788580663909, 'colsample_bytree': 0.6498244461726156}.
Best is trial 0 with value: 0.8730331625458602.
[I 2023-10-28 02:45:35,849] Trial 9 finished with value: 0.8769459031892352 and paramete
rs: {'learning_rate': 0.04918231750231449, 'n_estimators': 410, 'max_depth': 14, 'min_ch
ild_weight': 4, 'subsample': 0.8717928699303754, 'colsample_bytree': 0.591220659565479
2}. Best is trial 9 with value: 0.8769459031892352.
[I 2023-10-28 02:45:40,340] Trial 10 finished with value: 0.8666675080700813 and paramet
ers: {'learning_rate': 0.04140278516267972, 'n_estimators': 99, 'max_depth': 10, 'min_ch
ild_weight': 3, 'subsample': 0.9769989956071532, 'colsample_bytree': 0.655755516484153
5}. Best is trial 9 with value: 0.8769459031892352.
[I 2023-10-28 02:46:05,288] Trial 11 finished with value: 0.8733442904293595 and paramet
ers: {'learning_rate': 0.03889668442500809, 'n_estimators': 368, 'max_depth': 15, 'min_c
hild_weight': 2, 'subsample': 0.5124106921456655, 'colsample_bytree': 0.761816182206937
9}. Best is trial 9 with value: 0.8769459031892352.
[I 2023-10-28 02:46:25,801] Trial 12 finished with value: 0.877962994548071 and paramete
rs: {'learning_rate': 0.03954111712828176, 'n_estimators': 318, 'max_depth': 14, 'min_ch
ild_weight': 2, 'subsample': 0.8128871664563415, 'colsample_bytree': 0.735110157786542
4}. Best is trial 12 with value: 0.877962994548071.
[I 2023-10-28 02:46:42,912] Trial 13 finished with value: 0.8766228551058577 and paramet
ers: {'learning_rate': 0.03499127550288929, 'n_estimators': 297, 'max_depth': 12, 'min_c
hild_weight': 1, 'subsample': 0.8455882276028399, 'colsample_bytree': 0.674389712467129
2}. Best is trial 12 with value: 0.877962994548071.
[I 2023-10-28 02:46:58,093] Trial 14 finished with value: 0.8760963503792812 and paramet
ers: {'learning_rate': 0.05036279805966896, 'n_estimators': 322, 'max_depth': 12, 'min_c
hild_weight': 3, 'subsample': 0.8060310021467934, 'colsample_bytree': 0.580588652372347
8}. Best is trial 12 with value: 0.877962994548071.
[I 2023-10-28 02:47:04,613] Trial 15 finished with value: 0.8611393824010645 and paramet
ers: {'learning_rate': 0.026108061750112724, 'n_estimators': 205, 'max_depth': 8, 'min_c
hild_weight': 3, 'subsample': 0.8844761402775875, 'colsample_bytree': 0.701785528985724
5}. Best is trial 12 with value: 0.877962994548071.
[I 2023-10-28 02:47:27,287] Trial 16 finished with value: 0.878178372107033 and paramete
rs: {'learning_rate': 0.06398653796636893, 'n_estimators': 413, 'max_depth': 13, 'min_ch
ild_weight': 2, 'subsample': 0.9284793865240664, 'colsample_bytree': 0.609949810369020
2}. Best is trial 16 with value: 0.878178372107033.
[I 2023-10-28 02:47:36,766] Trial 17 finished with value: 0.8750074750914104 and paramet
ers: {'learning_rate': 0.06480096222200592, 'n_estimators': 239, 'max_depth': 10, 'min_c
hild_weight': 2, 'subsample': 0.9473262532993952, 'colsample_bytree': 0.810629171847373
6}. Best is trial 16 with value: 0.878178372107033.
[I 2023-10-28 02:47:39,322] Trial 18 finished with value: 0.6279779354243878 and paramet
ers: {'learning_rate': 0.0020331260522063258, 'n_estimators': 63, 'max_depth': 8, 'min_c
hild_weight': 1, 'subsample': 0.9331988539166145, 'colsample_bytree': 0.708303471461670
4}. Best is trial 16 with value: 0.878178372107033.
[I 2023-10-28 02:47:59,791] Trial 19 finished with value: 0.8777954653056931 and paramet
ers: {'learning_rate': 0.07351751808885283, 'n_estimators': 425, 'max_depth': 12, 'min_c
hild_weight': 2, 'subsample': 0.9993217686480154, 'colsample_bytree': 0.502810282425774
3}. Best is trial 16 with value: 0.878178372107033.
[I 2023-10-28 02:48:17,297] Trial 20 finished with value: 0.8767305492544757 and paramet
ers: {'learning_rate': 0.029799559049602256, 'n_estimators': 335, 'max_depth': 13, 'min_
child_weight': 3, 'subsample': 0.7859946148094156, 'colsample_bytree': 0.633037265205229
5}. Best is trial 16 with value: 0.878178372107033.
[I 2023-10-28 02:48:35,711] Trial 21 finished with value: 0.8774604433310683 and paramet
ers: {'learning_rate': 0.07205501603670462, 'n_estimators': 419, 'max_depth': 11, 'min_c
hild_weight': 2, 'subsample': 0.9706821712320975, 'colsample_bytree': 0.511995620713129
5}. Best is trial 16 with value: 0.878178372107033.
[I 2023-10-28 02:49:03,064] Trial 22 finished with value: 0.8787168407024678 and paramet
ers: {'learning_rate': 0.06541521540352564, 'n_estimators': 446, 'max_depth': 14, 'min_c
hild_weight': 2, 'subsample': 0.9971089033130367, 'colsample_bytree': 0.551110867283939
```

```
8}. Best is trial 22 with value: 0.8787168407024678.
[I 2023-10-28 02:49:29,962] Trial 23 finished with value: 0.8779629966957258 and paramet
ers: {'learning_rate': 0.04805898382134272, 'n_estimators': 378, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.915728984517843, 'colsample_bytree': 0.562079927646052
4}. Best is trial 22 with value: 0.8787168407024678.
[I 2023-10-28 02:50:04,606] Trial 24 finished with value: 0.8787407849055378 and paramet
ers: {'learning_rate': 0.0573162060051404, 'n_estimators': 451, 'max_depth': 15, 'min_ch
ild_weight': 1, 'subsample': 0.9122672302110972, 'colsample_bytree': 0.558973758503251
9}. Best is trial 24 with value: 0.8787407849055378.
[I 2023-10-28 02:50:39,159] Trial 25 finished with value: 0.878333903833961 and paramete
rs: {'learning_rate': 0.06108807197575342, 'n_estimators': 453, 'max_depth': 15, 'min_ch
ild_weight': 1, 'subsample': 0.9107686312962465, 'colsample_bytree': 0.609596392703109
2}. Best is trial 24 with value: 0.8787407849055378.
[I 2023-10-28 02:51:14,423] Trial 26 finished with value: 0.8790757961418885 and paramet
ers: {'learning_rate': 0.056778204181094506, 'n_estimators': 460, 'max_depth': 15, 'min_
child_weight': 1, 'subsample': 0.8985721277821027, 'colsample_bytree': 0.549431297254727
7}. Best is trial 26 with value: 0.8790757961418885.
[I 2023-10-28 02:51:53,889] Trial 27 finished with value: 0.8793390291762838 and paramet
ers: {'learning_rate': 0.053635716307868196, 'n_estimators': 500, 'max_depth': 15, 'min_
child_weight': 1, 'subsample': 0.9984222105952384, 'colsample_bytree': 0.550055136202684
9}. Best is trial 27 with value: 0.8793390291762838.
[I 2023-10-28 02:52:31,391] Trial 28 finished with value: 0.8787048653794507 and paramet
ers: {'learning_rate': 0.053298328777500424, 'n_estimators': 493, 'max_depth': 15, 'min_
child_weight': 1, 'subsample': 0.8971293800089727, 'colsample_bytree': 0.547442203355428
6}. Best is trial 27 with value: 0.8793390291762838.
[I 2023-10-28 02:53:10,815] Trial 29 finished with value: 0.8788723752929355 and paramet
ers: {'learning_rate': 0.05471055242693864, 'n_estimators': 500, 'max_depth': 15, 'min_c
hild_weight': 1, 'subsample': 0.9550221742366631, 'colsample_bytree': 0.628404272583964
3}. Best is trial 27 with value: 0.8793390291762838.
[I 2023-10-28 02:53:41,146] Trial 30 finished with value: 0.8783578516164556 and paramet
ers: {'learning_rate': 0.04609796872807228, 'n_estimators': 494, 'max_depth': 13, 'min_c
hild_weight': 1, 'subsample': 0.956477088886316, 'colsample_bytree': 0.632923583042904
4}. Best is trial 27 with value: 0.8793390291762838.
[I 2023-10-28 02:54:17,089] Trial 31 finished with value: 0.8790757768129955 and paramet
ers: {'learning_rate': 0.05516148739206619, 'n_estimators': 457, 'max_depth': 15, 'min_c
hild_weight': 1, 'subsample': 0.9497722757955012, 'colsample_bytree': 0.570281908554998
2}. Best is trial 27 with value: 0.8793390291762838.
[I 2023-10-28 02:54:53,149] Trial 32 finished with value: 0.8792792112637621 and paramet
ers: {'learning_rate': 0.05420490177545284, 'n_estimators': 466, 'max_depth': 15, 'min_c
hild_weight': 1, 'subsample': 0.9540283667551148, 'colsample_bytree': 0.529950304470081
3}. Best is trial 27 with value: 0.8793390291762838.
[I 2023-10-28 02:55:21,160] Trial 33 finished with value: 0.8793749415435217 and paramet
ers: {'learning_rate': 0.04450156823730534, 'n_estimators': 390, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.978426484861522, 'colsample_bytree': 0.501547999301231}.
Best is trial 33 with value: 0.8793749415435217.
[I 2023-10-28 02:55:38,933] Trial 34 finished with value: 0.8768741364414385 and paramet
ers: {'learning_rate': 0.04308161529279257, 'n_estimators': 389, 'max_depth': 11, 'min_c
hild_weight': 2, 'subsample': 0.9824296820390607, 'colsample_bytree': 0.502643038800216
7}. Best is trial 33 with value: 0.8793749415435217.
[I 2023-10-28 02:56:05,793] Trial 35 finished with value: 0.8780587219642909 and paramet
ers: {'learning_rate': 0.04610382267111903, 'n_estimators': 471, 'max_depth': 14, 'min_c
hild_weight': 3, 'subsample': 0.9979646276082563, 'colsample_bytree': 0.53948489679946
5}. Best is trial 33 with value: 0.8793749415435217.
[I 2023-10-28 02:56:21,863] Trial 36 finished with value: 0.8762997862618176 and paramet
ers: {'learning_rate': 0.05122375432552635, 'n_estimators': 360, 'max_depth': 13, 'min_c
hild_weight': 7, 'subsample': 0.8949364253121559, 'colsample_bytree': 0.534726756924247
4}. Best is trial 33 with value: 0.8793749415435217.
[I 2023-10-28 02:56:31,448] Trial 37 finished with value: 0.8703768264327337 and paramet
ers: {'learning_rate': 0.08140946029266499, 'n_estimators': 403, 'max_depth': 6, 'min_ch
ild_weight': 1, 'subsample': 0.948604026987979, 'colsample_bytree': 0.5303032301382944}.
```

```
Best is trial 33 with value: 0.8793749415435217.
[I 2023-10-28 02:57:05,572] Trial 38 finished with value: 0.8795065612822015 and paramet
ers: {'learning_rate': 0.03517223355118692, 'n_estimators': 472, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.8752967918966579, 'colsample_bytree': 0.58524922084730
8}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 02:57:19,053] Trial 39 finished with value: 0.8729972788140193 and paramet
ers: {'learning_rate': 0.03285734854318663, 'n_estimators': 434, 'max_depth': 9, 'min_ch
ild_weight': 6, 'subsample': 0.8502456309555955, 'colsample_bytree': 0.592150914293561
3}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 02:57:30,039] Trial 40 finished with value: 0.8639752259431257 and paramet
ers: {'learning_rate': 0.028353700918383396, 'n_estimators': 478, 'max_depth': 6, 'min_c
hild_weight': 6, 'subsample': 0.9280543517393911, 'colsample_bytree': 0.501847932080106
3}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 02:58:03,337] Trial 41 finished with value: 0.8792193833288513 and paramet
ers: {'learning_rate': 0.04397117438790259, 'n_estimators': 469, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.8682245249202538, 'colsample_bytree': 0.536395378825464
4}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 02:58:32,362] Trial 42 finished with value: 0.8781065853144583 and paramet
ers: {'learning_rate': 0.042812007774456685, 'n_estimators': 476, 'max_depth': 14, 'min_
child_weight': 2, 'subsample': 0.8694411345812473, 'colsample_bytree': 0.523820955448592
4}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 02:58:59,171] Trial 43 finished with value: 0.8786928793181596 and paramet
ers: {'learning_rate': 0.036742267890326925, 'n_estimators': 435, 'max_depth': 13, 'min_
child_weight': 1, 'subsample': 0.9792326386410376, 'colsample_bytree': 0.575723483923161
1}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 02:59:19,522] Trial 44 finished with value: 0.8762997862618176 and paramet
ers: {'learning_rate': 0.021106091304270307, 'n_estimators': 398, 'max_depth': 12, 'min_
child_weight': 2, 'subsample': 0.8734890696196611, 'colsample_bytree': 0.520728301513175
7}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 02:59:54,863] Trial 45 finished with value: 0.8789800479650058 and paramet
ers: {'learning_rate': 0.03740199396504247, 'n_estimators': 498, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.9651392401023553, 'colsample_bytree': 0.577451983178497
8}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:00:17,025] Trial 46 finished with value: 0.8778911948695678 and paramet
ers: {'learning_rate': 0.045181333167773, 'n_estimators': 472, 'max_depth': 11, 'min_chi
ld_weight': 1, 'subsample': 0.9314674616259622, 'colsample_bytree': 0.5303880353086554}.
Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:00:23,607] Trial 47 finished with value: 0.8543429205477622 and paramet
ers: {'learning_rate': 0.04048704259435101, 'n_estimators': 356, 'max_depth': 4, 'min_ch
ild_weight': 5, 'subsample': 0.8306960999320847, 'colsample_bytree': 0.599244618249335
6}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:00:48,208] Trial 48 finished with value: 0.8777476069667202 and paramet
ers: {'learning_rate': 0.0498066138195552, 'n_estimators': 433, 'max_depth': 14, 'min_ch
ild_weight': 3, 'subsample': 0.8593031321815624, 'colsample_bytree': 0.566623482217359
3}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:00:58,213] Trial 49 finished with value: 0.8735477055512331 and paramet
ers: {'learning_rate': 0.03190716143802934, 'n_estimators': 141, 'max_depth': 13, 'min_c
hild_weight': 1, 'subsample': 0.9724320192433872, 'colsample_bytree': 0.520908631548357
9}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:01:15,543] Trial 50 finished with value: 0.8767664301227768 and paramet
ers: {'learning_rate': 0.03667788154792392, 'n_estimators': 378, 'max_depth': 12, 'min_c
hild_weight': 4, 'subsample': 0.8823987971075893, 'colsample_bytree': 0.500162275905215
7}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:01:50,334] Trial 51 finished with value: 0.8779869351717163 and paramet
ers: {'learning_rate': 0.05822249427701896, 'n_estimators': 444, 'max_depth': 15, 'min_c
hild_weight': 1, 'subsample': 0.908730658067507, 'colsample_bytree': 0.550415029074901
8}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:02:21,072] Trial 52 finished with value: 0.8790279363711457 and paramet
ers: {'learning_rate': 0.052124203926620594, 'n_estimators': 465, 'max_depth': 15, 'min_
child_weight': 2, 'subsample': 0.936284951574015, 'colsample_bytree': 0.541734259212183
```

```
7}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:02:40,982] Trial 53 finished with value: 0.8788365015834838 and paramet
ers: {'learning_rate': 0.04303283070435664, 'n_estimators': 272, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.8947568598986525, 'colsample_bytree': 0.589237355234367
5}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:03:12,819] Trial 54 finished with value: 0.8769818148405879 and paramet
ers: {'learning_rate': 0.04850840745219526, 'n_estimators': 482, 'max_depth': 15, 'min_c
hild_weight': 2, 'subsample': 0.8445944536781209, 'colsample_bytree': 0.560078804977147
8}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:03:38,089] Trial 55 finished with value: 0.8790159510257396 and paramet
ers: {'learning_rate': 0.05717129038713601, 'n_estimators': 414, 'max_depth': 13, 'min_c
hild_weight': 1, 'subsample': 0.9827569730399598, 'colsample_bytree': 0.520898210292616
5}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:04:07,664] Trial 56 finished with value: 0.8785732334707271 and paramet
ers: {'learning_rate': 0.04018998395487645, 'n_estimators': 463, 'max_depth': 14, 'min_c
hild_weight': 2, 'subsample': 0.9396064596168003, 'colsample_bytree': 0.616236883635389
3}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:04:32,817] Trial 57 finished with value: 0.8788364937087497 and paramet
ers: {'learning_rate': 0.05337496900245027, 'n_estimators': 483, 'max_depth': 12, 'min_c
hild_weight': 1, 'subsample': 0.9647125108249168, 'colsample_bytree': 0.547333295433880
8}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:04:45,522] Trial 58 finished with value: 0.871681064245983 and paramete
rs: {'learning_rate': 0.06174478988221491, 'n_estimators': 448, 'max_depth': 7, 'min_chi
ld_weight': 1, 'subsample': 0.7818045268309332, 'colsample_bytree': 0.5824439117921977}.
Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:04:51,941] Trial 59 finished with value: 0.8427243723845816 and paramet
ers: {'learning_rate': 0.04664868638937735, 'n_estimators': 427, 'max_depth': 2, 'min_ch
ild_weight': 2, 'subsample': 0.9183683372247654, 'colsample_bytree': 0.656351503778926
5}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:05:18,257] Trial 60 finished with value: 0.8792313579359835 and paramet
ers: {'learning_rate': 0.03445184746027906, 'n_estimators': 299, 'max_depth': 15, 'min_c
hild_weight': 1, 'subsample': 0.8903171369141898, 'colsample_bytree': 0.562164498223149
8}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:05:42,749] Trial 61 finished with value: 0.8786210903779302 and paramet
ers: {'learning_rate': 0.03238788882066122, 'n_estimators': 269, 'max_depth': 15, 'min_c
hild_weight': 1, 'subsample': 0.8845914453107333, 'colsample_bytree': 0.572320084511790
1}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:06:07,096] Trial 62 finished with value: 0.8786928936358583 and paramet
ers: {'learning_rate': 0.035873706338980745, 'n_estimators': 330, 'max_depth': 14, 'min_
child_weight': 1, 'subsample': 0.8986511433187594, 'colsample_bytree': 0.555784424464007
8}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:06:34,125] Trial 63 finished with value: 0.8784296505790739 and paramet
ers: {'learning_rate': 0.02586316974549273, 'n_estimators': 304, 'max_depth': 15, 'min_c
hild_weight': 1, 'subsample': 0.8603124481658615, 'colsample_bytree': 0.537545716956709
5}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:06:49,847] Trial 64 finished with value: 0.8780587255437154 and paramet
ers: {'learning_rate': 0.03915556525437841, 'n_estimators': 203, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.9217929276634103, 'colsample_bytree': 0.601202162394251
6}. Best is trial 38 with value: 0.8795065612822015.
[I 2023-10-28 03:07:08,705] Trial 65 finished with value: 0.8796381667031827 and paramet
ers: {'learning_rate': 0.048877025577590806, 'n_estimators': 255, 'max_depth': 15, 'min_
child_weight': 2, 'subsample': 0.9515379458177781, 'colsample_bytree': 0.511487038615984
7}. Best is trial 65 with value: 0.8796381667031827.
[I 2023-10-28 03:07:23,449] Trial 66 finished with value: 0.8781903445665102 and paramet
ers: {'learning_rate': 0.04951173856553722, 'n_estimators': 256, 'max_depth': 13, 'min_c
hild_weight': 2, 'subsample': 0.9502195169821862, 'colsample_bytree': 0.512955554692677
2}. Best is trial 65 with value: 0.8796381667031827.
[I 2023-10-28 03:07:41,038] Trial 67 finished with value: 0.879087777191985 and paramete
rs: {'learning_rate': 0.0423117977268075, 'n_estimators': 226, 'max_depth': 15, 'min_ch
ild_weight': 2, 'subsample': 0.9914553322951167, 'colsample_bytree': 0.514770402655304
```

```
6}. Best is trial 65 with value: 0.8796381667031827.
[I 2023-10-28 03:08:02,708] Trial 68 finished with value: 0.8791954391257815 and paramet
ers: {'learning_rate': 0.04547405851816248, 'n_estimators': 291, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.9635571627088411, 'colsample_bytree': 0.53435337985161
9}. Best is trial 65 with value: 0.8796381667031827.
[I 2023-10-28 03:08:15,148] Trial 69 finished with value: 0.8775920702285974 and paramet
ers: {'learning_rate': 0.052349330216008266, 'n_estimators': 242, 'max_depth': 13, 'min_
child_weight': 4, 'subsample': 0.9849504926963643, 'colsample_bytree': 0.567179028072641
5}. Best is trial 65 with value: 0.8796381667031827.
[I 2023-10-28 03:08:30,291] Trial 70 finished with value: 0.8779749856205564 and paramet
ers: {'learning_rate': 0.03894963108545073, 'n_estimators': 185, 'max_depth': 15, 'min_c
hild_weight': 2, 'subsample': 0.9421872217448886, 'colsample_bytree': 0.500885743489746
5}. Best is trial 65 with value: 0.8796381667031827.
[I 2023-10-28 03:08:52,109] Trial 71 finished with value: 0.8796980204099508 and paramet
ers: {'learning_rate': 0.04493942651894983, 'n_estimators': 296, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.9653175949139311, 'colsample_bytree': 0.534394112599438
1}. Best is trial 71 with value: 0.8796980204099508.
[I 2023-10-28 03:09:15,832] Trial 72 finished with value: 0.8796381824526511 and paramet
ers: {'learning_rate': 0.04866639539374388, 'n_estimators': 319, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.9973144902244658, 'colsample_bytree': 0.530834312091193
3}. Best is trial 71 with value: 0.8796980204099508.
[I 2023-10-28 03:09:39,199] Trial 73 finished with value: 0.8791834924381611 and paramet
ers: {'learning_rate': 0.04979324447687668, 'n_estimators': 311, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.9965470701655165, 'colsample_bytree': 0.521194663764590
7}. Best is trial 71 with value: 0.8796980204099508.
[I 2023-10-28 03:10:03,884] Trial 74 finished with value: 0.8785134177058602 and paramet
ers: {'learning_rate': 0.046666964626888435, 'n_estimators': 288, 'max_depth': 15, 'min_
child_weight': 1, 'subsample': 0.9735442339398943, 'colsample_bytree': 0.558647599747449
5}. Best is trial 71 with value: 0.8796980204099508.
[I 2023-10-28 03:10:32,218] Trial 75 finished with value: 0.8800330638611232 and paramet
ers: {'learning_rate': 0.03421951835454458, 'n_estimators': 329, 'max_depth': 15, 'min_c
hild_weight': 1, 'subsample': 0.9576080984363367, 'colsample_bytree': 0.589856504476964
9}. Best is trial 75 with value: 0.8800330638611232.
[I 2023-10-28 03:10:53,964] Trial 76 finished with value: 0.8784775110657017 and paramet
ers: {'learning_rate': 0.05954823274560289, 'n_estimators': 338, 'max_depth': 13, 'min_c
hild_weight': 1, 'subsample': 0.963550700331536, 'colsample_bytree': 0.588222428312151
3}. Best is trial 75 with value: 0.8800330638611232.
[I 2023-10-28 03:11:14,753] Trial 77 finished with value: 0.8793749322370177 and paramet
ers: {'learning_rate': 0.04143360621151371, 'n_estimators': 318, 'max_depth': 14, 'min_c
hild_weight': 2, 'subsample': 0.9860881841891413, 'colsample_bytree': 0.51368878937138
2}. Best is trial 75 with value: 0.8800330638611232.
[I 2023-10-28 03:11:31,240] Trial 78 finished with value: 0.8781783799817673 and paramet
ers: {'learning_rate': 0.043509408912890686, 'n_estimators': 352, 'max_depth': 12, 'min_
child_weight': 3, 'subsample': 0.9825388181364458, 'colsample_bytree': 0.508528379211525
2}. Best is trial 75 with value: 0.8800330638611232.
[I 2023-10-28 03:11:51,865] Trial 79 finished with value: 0.8792911851550093 and paramet
ers: {'learning_rate': 0.04125119570643407, 'n_estimators': 316, 'max_depth': 14, 'min_c
hild_weight': 2, 'subsample': 0.9905255074134165, 'colsample_bytree': 0.546520374121824
9}. Best is trial 75 with value: 0.8800330638611232.
[I 2023-10-28 03:12:04,443] Trial 80 finished with value: 0.8760604659315554 and paramet
ers: {'learning_rate': 0.04827728734709924, 'n_estimators': 284, 'max_depth': 11, 'min_c
hild_weight': 2, 'subsample': 0.9751791523937877, 'colsample_bytree': 0.529348601770335
6}. Best is trial 75 with value: 0.8800330638611232.
[I 2023-10-28 03:12:24,073] Trial 81 finished with value: 0.8786450238427264 and paramet
ers: {'learning_rate': 0.04067877504556524, 'n_estimators': 322, 'max_depth': 14, 'min_c
hild_weight': 3, 'subsample': 0.9968345508065137, 'colsample_bytree': 0.54676242272874
5}. Best is trial 75 with value: 0.8800330638611232.
[I 2023-10-28 03:12:44,787] Trial 82 finished with value: 0.8785014531211168 and paramet
ers: {'learning_rate': 0.03787667802304684, 'n_estimators': 314, 'max_depth': 14, 'min_c
hild_weight': 2, 'subsample': 0.9586614369960118, 'colsample_bytree': 0.513244980743448
```

```
2}. Best is trial 75 with value: 0.8800330638611232.
[I 2023-10-28 03:13:04,427] Trial 83 finished with value: 0.8783698369618615 and paramet
ers: {'learning_rate': 0.03463873833305672, 'n_estimators': 344, 'max_depth': 13, 'min_c
hild_weight': 2, 'subsample': 0.9981204973697293, 'colsample_bytree': 0.576682240239120
6}. Best is trial 75 with value: 0.8800330638611232.
[I 2023-10-28 03:13:19,434] Trial 84 finished with value: 0.8779989040517691 and paramet
ers: {'learning_rate': 0.04147818787653034, 'n_estimators': 259, 'max_depth': 13, 'min_c
hild_weight': 2, 'subsample': 0.9820198661444905, 'colsample_bytree': 0.539922098035264
9}. Best is trial 75 with value: 0.8800330638611232.
[I 2023-10-28 03:13:38,648] Trial 85 finished with value: 0.8787886296426972 and paramet
ers: {'learning_rate': 0.05116546620692981, 'n_estimators': 329, 'max_depth': 14, 'min_c
hild_weight': 3, 'subsample': 0.9434905846521935, 'colsample_bytree': 0.552103275478586
2}. Best is trial 75 with value: 0.8800330638611232.
[I 2023-10-28 03:14:04,984] Trial 86 finished with value: 0.8795065684410506 and paramet
ers: {'learning_rate': 0.0462162127003814, 'n_estimators': 309, 'max_depth': 15, 'min_ch
ild_weight': 1, 'subsample': 0.9680660364681785, 'colsample_bytree': 0.513507638867391}.
Best is trial 75 with value: 0.8800330638611232.
[I 2023-10-28 03:14:35,140] Trial 87 finished with value: 0.8802244743086977 and paramet
ers: {'learning_rate': 0.05562297903073267, 'n_estimators': 381, 'max_depth': 15, 'min_c
hild_weight': 1, 'subsample': 0.9714353280416418, 'colsample_bytree': 0.52393471318861
6}. Best is trial 87 with value: 0.8802244743086977.
[I 2023-10-28 03:15:06,372] Trial 88 finished with value: 0.8791954570229045 and paramet
ers: {'learning_rate': 0.04441823346697751, 'n_estimators': 376, 'max_depth': 15, 'min_c
hild_weight': 1, 'subsample': 0.9295741758494145, 'colsample_bytree': 0.513303921301683
8}. Best is trial 87 with value: 0.8802244743086977.
[I 2023-10-28 03:15:26,619] Trial 89 finished with value: 0.8774604719664655 and paramet
ers: {'learning_rate': 0.05598163686736575, 'n_estimators': 367, 'max_depth': 15, 'min_c
hild_weight': 5, 'subsample': 0.969556693691313, 'colsample_bytree': 0.526518304625734
1}. Best is trial 87 with value: 0.8802244743086977.
[I 2023-10-28 03:15:55,259] Trial 90 finished with value: 0.8789920397533763 and paramet
ers: {'learning_rate': 0.04683616413513437, 'n_estimators': 389, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.9523311248585273, 'colsample_bytree': 0.500581690072672
2}. Best is trial 87 with value: 0.8802244743086977.
[I 2023-10-28 03:16:18,905] Trial 91 finished with value: 0.8790997303225696 and paramet
ers: {'learning_rate': 0.054484169419070916, 'n_estimators': 283, 'max_depth': 15, 'min_
child_weight': 1, 'subsample': 0.9715445030486217, 'colsample_bytree': 0.525820746971012
9}. Best is trial 87 with value: 0.8802244743086977.
[I 2023-10-28 03:16:44,302] Trial 92 finished with value: 0.8793270867839732 and paramet
ers: {'learning_rate': 0.0520750808961457, 'n_estimators': 303, 'max_depth': 15, 'min_ch
ild_weight': 1, 'subsample': 0.9568899578770158, 'colsample_bytree': 0.512829189564534
3}. Best is trial 87 with value: 0.8802244743086977.
[I 2023-10-28 03:16:56,019] Trial 93 finished with value: 0.8744690551761505 and paramet
ers: {'learning_rate': 0.04448372909369904, 'n_estimators': 348, 'max_depth': 9, 'min_ch
ild_weight': 1, 'subsample': 0.9862661526617608, 'colsample_bytree': 0.533817525110806
8}. Best is trial 87 with value: 0.8802244743086977.
[I 2023-10-28 03:17:18,529] Trial 94 finished with value: 0.8790638451589589 and paramet
ers: {'learning_rate': 0.04776917088880735, 'n_estimators': 262, 'max_depth': 15, 'min_c
hild_weight': 1, 'subsample': 0.9402019474220233, 'colsample_bytree': 0.569970574702186
7}. Best is trial 87 with value: 0.8802244743086977.
[I 2023-10-28 03:17:36,479] Trial 95 finished with value: 0.8788604071287678 and paramet
ers: {'learning_rate': 0.04973819750905015, 'n_estimators': 233, 'max_depth': 14, 'min_c
hild_weight': 1, 'subsample': 0.9993748578973978, 'colsample_bytree': 0.519125100158832
8}. Best is trial 87 with value: 0.8802244743086977.
[I 2023-10-28 03:17:56,774] Trial 96 finished with value: 0.8795544081670157 and paramet
ers: {'learning_rate': 0.05472337181088019, 'n_estimators': 325, 'max_depth': 13, 'min_c
hild_weight': 1, 'subsample': 0.9733826126946591, 'colsample_bytree': 0.543723659396644
8}. Best is trial 87 with value: 0.8802244743086977.
[I 2023-10-28 03:18:19,924] Trial 97 finished with value: 0.8786330699962569 and paramet
ers: {'learning_rate': 0.03746857726433674, 'n_estimators': 363, 'max_depth': 13, 'min_c
hild_weight': 1, 'subsample': 0.9255824457045603, 'colsample_bytree': 0.50793036720272}.
```

```
Best is trial 87 with value: 0.8802244743086977.
[I 2023-10-28 03:18:44,192] Trial 98 finished with value: 0.8794586943526094 and paramet
ers: {'learning_rate': 0.056302305578548734, 'n_estimators': 338, 'max_depth': 14, 'min_
child_weight': 1, 'subsample': 0.9613994259168135, 'colsample_bytree': 0.539159116355823
4}. Best is trial 87 with value: 0.8802244743086977.
[I 2023-10-28 03:19:06,121] Trial 99 finished with value: 0.8788005999545199 and paramet
ers: {'learning_rate': 0.058015258068739535, 'n_estimators': 339, 'max_depth': 13, 'min_
child_weight': 1, 'subsample': 0.9599637655180461, 'colsample_bytree': 0.558620370469330
9}. Best is trial 87 with value: 0.8802244743086977.
Best parameters: {'learning_rate': 0.05562297903073267, 'n_estimators': 381, 'max_dept
h': 15, 'min_child_weight': 1, 'subsample': 0.9714353280416418, 'colsample_bytree': 0.52
3934713188616}
```

In [66]:

```
#Printing the best hyperparameters and their corresponding cross-validation score
best_score = study.best_value
print(f"Best cross-validation score: {best_score}")
```

```
Best cross-validation score: 0.8802244743086977
```

Week 7 (Logistic)

In [67]:

```
import optuna
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
```

In [68]:

```
#Define Objective Function for Optuna
def objective(trial):
    # Hyperparameters to be tuned
    C = trial.suggest_float("C", 1e-5, 100, log=True)
    penalty = trial.suggest_categorical("penalty", ["l1", "l2"])
    solver = trial.suggest_categorical("solver", ["liblinear", "saga"])  # suitable for

    # Model
    lr = LogisticRegression(C=C, penalty=penalty, solver=solver, max_iter=5000, random_

    # Return cross-validation mean score
    return cross_val_score(lr, xtrain, ytrain, n_jobs=-1, cv=3).mean()
```

In [69]:

```
#Optuna Study for Each Variation
results = []

for i in range(3):  # Three variations
    study = optuna.create_study(direction="maximize")
    study.optimize(objective, n_trials=50)  # for example, 50 trials for each variation

    best_params = study.best_params
    best_score = study.best_value

    results.append({
        "Variation": i + 1,
        "Best Parameters": best_params,
        "Best Cross-Validation Score": best_score
    })
```

```
[I 2023-10-28 03:19:06,293] A new study created in memory with name: no-name-87a0118d-67
5b-4ca4-9307-ceb47b8ace9a
[I 2023-10-28 03:20:29,237] Trial 0 finished with value: 0.6265899334185699 and paramete
rs: {'C': 0.002716846584514881, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 0 with
value: 0.6265899334185699.
[I 2023-10-28 03:21:49,316] Trial 1 finished with value: 0.6265899334185699 and paramete
rs: {'C': 0.016799413407415493, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 0 with
value: 0.6265899334185699.
[I 2023-10-28 03:21:53,628] Trial 2 finished with value: 0.7958072395104008 and paramete
rs: {'C': 0.013316695869033626, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 2
with value: 0.7958072395104008.
[I 2023-10-28 03:23:13,259] Trial 3 finished with value: 0.6266377947642323 and paramete
rs: {'C': 5.2421065594233196e-05, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 2 wi
th value: 0.7958072395104008.
[I 2023-10-28 03:24:28,862] Trial 4 finished with value: 0.6265779675452406 and paramete
rs: {'C': 1.908894878400505e-05, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 2 wit
h value: 0.7958072395104008.
[I 2023-10-28 03:24:29,753] Trial 5 finished with value: 0.7697581706965274 and paramete
rs: {'C': 0.0015905261623133365, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial
2 with value: 0.7958072395104008.
[I 2023-10-28 03:26:02,912] Trial 6 finished with value: 0.6265899334185699 and paramete
rs: {'C': 2.871341000110989, 'penalty': 'l1', 'solver': 'saga'}. Best is trial 2 with va
lue: 0.7958072395104008.
[I 2023-10-28 03:26:16,150] Trial 7 finished with value: 0.7964773374369897 and paramete
rs: {'C': 85.18708010790898, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 7 wi
th value: 0.7964773374369897.
[I 2023-10-28 03:27:53,800] Trial 8 finished with value: 0.6266018988623684 and paramete
rs: {'C': 0.1481515107021096, 'penalty': 'l1', 'solver': 'saga'}. Best is trial 7 with v
alue: 0.7964773374369897.
[I 2023-10-28 03:29:15,101] Trial 9 finished with value: 0.6265899334185699 and paramete
rs: {'C': 0.040862871312964955, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 7 with
value: 0.7964773374369897.
[I 2023-10-28 03:29:27,909] Trial 10 finished with value: 0.7964773374369897 and paramet
ers: {'C': 43.51325850540886, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 7 w
ith value: 0.7964773374369897.
[I 2023-10-28 03:29:40,701] Trial 11 finished with value: 0.7964773374369897 and paramet
ers: {'C': 32.585066623594145, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 7
with value: 0.7964773374369897.
[I 2023-10-28 03:29:53,902] Trial 12 finished with value: 0.7964773374369897 and paramet
ers: {'C': 76.43883509716115, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 7 w
ith value: 0.7964773374369897.
[I 2023-10-28 03:30:06,292] Trial 13 finished with value: 0.7964175093589362 and paramet
ers: {'C': 5.320550718154589, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 7 w
ith value: 0.7964773374369897.
[I 2023-10-28 03:30:18,697] Trial 14 finished with value: 0.7964773374369897 and paramet
ers: {'C': 62.04696334926438, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 7 w
ith value: 0.7964773374369897.
[I 2023-10-28 03:30:30,591] Trial 15 finished with value: 0.7964294748027344 and paramet
ers: {'C': 2.6705031374202832, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 7
with value: 0.7964773374369897.
[I 2023-10-28 03:30:50,411] Trial 16 finished with value: 0.7964773400141754 and paramet
ers: {'C': 0.5083855249228197, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:30:57,326] Trial 17 finished with value: 0.7964294773799202 and paramet
ers: {'C': 0.5486520283122146, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:31:09,030] Trial 18 finished with value: 0.796381615175196 and paramete
rs: {'C': 0.44635894062175535, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:31:21,598] Trial 19 finished with value: 0.7964653711341293 and paramet
```

```
ers: {'C': 10.04908662241622, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:31:34,315] Trial 20 finished with value: 0.7964534056903311 and paramet
ers: {'C': 11.946875282550316, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:31:47,401] Trial 21 finished with value: 0.7964653715636603 and paramet
ers: {'C': 19.598656920917968, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:32:00,614] Trial 22 finished with value: 0.7964773374369897 and paramet
ers: {'C': 99.76522178093373, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:32:08,845] Trial 23 finished with value: 0.7964773374369897 and paramet
ers: {'C': 1.0258448675185912, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:32:21,536] Trial 24 finished with value: 0.7964653715636603 and paramet
ers: {'C': 20.849443318770984, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:32:33,815] Trial 25 finished with value: 0.7964175093589362 and paramet
ers: {'C': 4.253983225296696, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:32:35,302] Trial 26 finished with value: 0.6597701352726731 and paramet
ers: {'C': 27.19173960313432, 'penalty': 'l2', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:32:48,227] Trial 27 finished with value: 0.7964773374369897 and paramet
ers: {'C': 98.53104645507123, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:32:56,124] Trial 28 finished with value: 0.7964653715636603 and paramet
ers: {'C': 1.199762322844465, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:32:57,702] Trial 29 finished with value: 0.6597701352726731 and paramet
ers: {'C': 7.015314942453924, 'penalty': 'l2', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:33:09,441] Trial 30 finished with value: 0.7964534056903311 and paramet
ers: {'C': 11.033847572411146, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:33:22,533] Trial 31 finished with value: 0.7964773374369897 and paramet
ers: {'C': 35.98125497973188, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:33:35,475] Trial 32 finished with value: 0.7964773374369897 and paramet
ers: {'C': 43.6242794391833, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16 w
ith value: 0.7964773400141754.
[I 2023-10-28 03:33:47,796] Trial 33 finished with value: 0.7964653715636603 and paramet
ers: {'C': 25.111209297161675, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:35:03,447] Trial 34 finished with value: 0.6265899334185699 and paramet
ers: {'C': 3.6827805406773444, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 16 with
value: 0.7964773400141754.
[I 2023-10-28 03:35:16,747] Trial 35 finished with value: 0.7964773374369897 and paramet
ers: {'C': 41.66749089466201, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:35:29,453] Trial 36 finished with value: 0.7964653711341293 and paramet
ers: {'C': 10.32823971133896, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:36:47,867] Trial 37 finished with value: 0.6265899334185699 and paramet
ers: {'C': 1.5714521663286394, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 16 with
value: 0.7964773400141754.
[I 2023-10-28 03:37:00,609] Trial 38 finished with value: 0.7964653715636603 and paramet
ers: {'C': 15.945782554452155, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:38:36,232] Trial 39 finished with value: 0.6266018988623684 and paramet
```

```
ers: {'C': 0.1805523678731965, 'penalty': 'l1', 'solver': 'saga'}. Best is trial 16 with
value: 0.7964773400141754.
[I 2023-10-28 03:38:37,608] Trial 40 finished with value: 0.6304428325230141 and paramet
ers: {'C': 6.893351482317751, 'penalty': 'l2', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:38:50,866] Trial 41 finished with value: 0.7964773374369897 and paramet
ers: {'C': 75.47057757275029, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:39:03,816] Trial 42 finished with value: 0.7964773374369897 and paramet
ers: {'C': 52.52176302947424, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:39:16,709] Trial 43 finished with value: 0.7964653715636603 and paramet
ers: {'C': 30.26640377053616, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:39:29,684] Trial 44 finished with value: 0.7964773374369897 and paramet
ers: {'C': 61.577593133099576, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:41:03,546] Trial 45 finished with value: 0.6265899334185699 and paramet
ers: {'C': 2.8688485493218883, 'penalty': 'l1', 'solver': 'saga'}. Best is trial 16 with
value: 0.7964773400141754.
[I 2023-10-28 03:41:16,802] Trial 46 finished with value: 0.7964773374369897 and paramet
ers: {'C': 95.6536013301367, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16 w
ith value: 0.7964773400141754.
[I 2023-10-28 03:41:29,662] Trial 47 finished with value: 0.7964534061198622 and paramet
ers: {'C': 16.32972289477676, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:41:42,503] Trial 48 finished with value: 0.7964175093589362 and paramet
ers: {'C': 6.179734989464446, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:41:55,075] Trial 49 finished with value: 0.7964773374369897 and paramet
ers: {'C': 33.21872833453844, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7964773400141754.
[I 2023-10-28 03:41:55,077] A new study created in memory with name: no-name-d206d3cc-18
55-480d-be41-62d887df82ca
[I 2023-10-28 03:41:56,482] Trial 0 finished with value: 0.6304428325230141 and paramete
rs: {'C': 0.0016821411406619299, 'penalty': 'l2', 'solver': 'liblinear'}. Best is trial
0 with value: 0.6304428325230141.
[I 2023-10-28 03:43:15,006] Trial 1 finished with value: 0.6265899334185699 and paramete
rs: {'C': 32.01095232905527, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 0 with va
lue: 0.6304428325230141.
[I 2023-10-28 03:43:35,169] Trial 2 finished with value: 0.796465374570377 and parameter
s: {'C': 0.5132863734151747, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 2 wi
th value: 0.796465374570377.
[I 2023-10-28 03:45:02,158] Trial 3 finished with value: 0.6265899334185699 and paramete
rs: {'C': 6.0115757018615925, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 2 with v
alue: 0.796465374570377.
[I 2023-10-28 03:46:21,741] Trial 4 finished with value: 0.6265899334185699 and paramete
rs: {'C': 0.028385304605215413, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 2 with
value: 0.796465374570377.
[I 2023-10-28 03:47:39,257] Trial 5 finished with value: 0.6266138647356975 and paramete
rs: {'C': 0.00014868321106692534, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 2 wi
th value: 0.796465374570377.
[I 2023-10-28 03:47:41,090] Trial 6 finished with value: 0.6696775227376742 and paramete
rs: {'C': 0.05647917582844463, 'penalty': 'l2', 'solver': 'liblinear'}. Best is trial 2
with value: 0.796465374570377.
[I 2023-10-28 03:49:00,197] Trial 7 finished with value: 0.6265899334185699 and paramete
rs: {'C': 0.09282020895633525, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 2 with
value: 0.796465374570377.
[I 2023-10-28 03:49:08,887] Trial 8 finished with value: 0.7964773374369897 and paramete
rs: {'C': 0.9448749353999538, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 8 w
```

```
ith value: 0.7964773374369897.
[I 2023-10-28 03:49:10,509] Trial 9 finished with value: 0.6614333332492311 and paramete
rs: {'C': 4.657838075286897e-05, 'penalty': 'l2', 'solver': 'liblinear'}. Best is trial
8 with value: 0.7964773374369897.
[I 2023-10-28 03:49:23,578] Trial 10 finished with value: 0.7964773374369897 and paramet
ers: {'C': 97.82955991535204, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 8 w
ith value: 0.7964773374369897.
[I 2023-10-28 03:49:36,019] Trial 11 finished with value: 0.7964773374369897 and paramet
ers: {'C': 46.443170694743344, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 8
with value: 0.7964773374369897.
[I 2023-10-28 03:49:48,125] Trial 12 finished with value: 0.7964414398170018 and paramet
ers: {'C': 2.2023412019717172, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 8
with value: 0.7964773374369897.
[I 2023-10-28 03:50:01,103] Trial 13 finished with value: 0.7964773374369897 and paramet
ers: {'C': 49.19582383494443, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 8 w
ith value: 0.7964773374369897.
[I 2023-10-28 03:50:12,715] Trial 14 finished with value: 0.7964055439151378 and paramet
ers: {'C': 3.639432884804646, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 8 w
ith value: 0.7964773374369897.
[I 2023-10-28 03:50:25,728] Trial 15 finished with value: 0.7964773374369897 and paramet
ers: {'C': 92.91802633150198, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 8 w
ith value: 0.7964773374369897.
[I 2023-10-28 03:50:44,766] Trial 16 finished with value: 0.7965132372046323 and paramet
ers: {'C': 0.5894325708616208, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 03:50:52,434] Trial 17 finished with value: 0.7963816134570721 and paramet
ers: {'C': 0.5170919408314429, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 03:50:56,520] Trial 18 finished with value: 0.7955559570295482 and paramet
ers: {'C': 0.010959753648981605, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial
16 with value: 0.7965132372046323.
[I 2023-10-28 03:51:10,775] Trial 19 finished with value: 0.7963337529704718 and paramet
ers: {'C': 0.31801513881415144, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 1
6 with value: 0.7965132372046323.
[I 2023-10-28 03:51:15,045] Trial 20 finished with value: 0.7908415360924136 and paramet
ers: {'C': 0.0053929281871386665, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial
16 with value: 0.7965132372046323.
[I 2023-10-28 03:51:27,873] Trial 21 finished with value: 0.7964414402465327 and paramet
ers: {'C': 7.345192530484432, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 03:51:40,969] Trial 22 finished with value: 0.7964534056903311 and paramet
ers: {'C': 11.775632422927051, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 03:51:52,936] Trial 23 finished with value: 0.7964653715636603 and paramet
ers: {'C': 1.4131476396084937, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 03:52:05,894] Trial 24 finished with value: 0.7964534061198622 and paramet
ers: {'C': 17.920926360917246, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 03:52:12,808] Trial 25 finished with value: 0.7963098190761585 and paramet
ers: {'C': 0.14659670273806485, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 1
6 with value: 0.7965132372046323.
[I 2023-10-28 03:53:50,178] Trial 26 finished with value: 0.6265899334185699 and paramet
ers: {'C': 1.258001023185168, 'penalty': 'l1', 'solver': 'saga'}. Best is trial 16 with
value: 0.7965132372046323.
[I 2023-10-28 03:54:02,954] Trial 27 finished with value: 0.7964653715636603 and paramet
ers: {'C': 15.725569887204054, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 03:54:15,205] Trial 28 finished with value: 0.7964055439151378 and paramet
ers: {'C': 3.423569742808199, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
```

```
with value: 0.7965132372046323.
[I 2023-10-28 03:54:23,401] Trial 29 finished with value: 0.7963098212238133 and paramet
ers: {'C': 0.3017581693910004, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 03:54:31,779] Trial 30 finished with value: 0.7964653715636603 and paramet
ers: {'C': 1.1347337777915463, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 03:54:45,192] Trial 31 finished with value: 0.7964773374369897 and paramet
ers: {'C': 69.32544071334452, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 03:54:58,041] Trial 32 finished with value: 0.7964653715636603 and paramet
ers: {'C': 32.240112713903024, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 03:55:11,247] Trial 33 finished with value: 0.7964653715636603 and paramet
ers: {'C': 25.822779629367247, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 03:56:48,626] Trial 34 finished with value: 0.6265899334185699 and paramet
ers: {'C': 9.245334475114877, 'penalty': 'l1', 'solver': 'saga'}. Best is trial 16 with
value: 0.7965132372046323.
[I 2023-10-28 03:57:01,473] Trial 35 finished with value: 0.7964773374369897 and paramet
ers: {'C': 61.21767913156288, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 03:58:19,203] Trial 36 finished with value: 0.6265899334185699 and paramet
ers: {'C': 4.86077872001674, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 16 with v
alue: 0.7965132372046323.
[I 2023-10-28 03:58:32,027] Trial 37 finished with value: 0.7964773374369897 and paramet
ers: {'C': 93.7651943057264, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16 w
ith value: 0.7965132372046323.
[I 2023-10-28 03:58:33,785] Trial 38 finished with value: 0.6715082356388158 and paramet
ers: {'C': 17.17402277176726, 'penalty': 'l2', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 04:00:11,214] Trial 39 finished with value: 0.6265899334185699 and paramet
ers: {'C': 6.495665994526917, 'penalty': 'l1', 'solver': 'saga'}. Best is trial 16 with
value: 0.7965132372046323.
[I 2023-10-28 04:00:12,930] Trial 40 finished with value: 0.6547685797649794 and paramet
ers: {'C': 32.08668830085159, 'penalty': 'l2', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 04:00:26,101] Trial 41 finished with value: 0.7964773374369897 and paramet
ers: {'C': 40.29774028639572, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 04:00:39,213] Trial 42 finished with value: 0.7964773374369897 and paramet
ers: {'C': 97.79322378016501, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 04:00:52,089] Trial 43 finished with value: 0.7964773374369897 and paramet
ers: {'C': 38.41446404095125, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 04:01:03,874] Trial 44 finished with value: 0.7964175093589362 and paramet
ers: {'C': 2.76514777366345, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16 w
ith value: 0.7965132372046323.
[I 2023-10-28 04:01:16,797] Trial 45 finished with value: 0.7964534056903311 and paramet
ers: {'C': 11.790535215460718, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 04:01:22,326] Trial 46 finished with value: 0.7957115309935977 and paramet
ers: {'C': 0.05578265759860116, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 1
6 with value: 0.7965132372046323.
[I 2023-10-28 04:02:42,546] Trial 47 finished with value: 0.6265899334185699 and paramet
ers: {'C': 0.6235419631291654, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 16 with
value: 0.7965132372046323.
[I 2023-10-28 04:02:55,207] Trial 48 finished with value: 0.7964175093589362 and paramet
ers: {'C': 6.278587706082513, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
```

```
with value: 0.7965132372046323.
[I 2023-10-28 04:03:06,951] Trial 49 finished with value: 0.7964414402465327 and paramet
ers: {'C': 2.4788940599957576, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 16
with value: 0.7965132372046323.
[I 2023-10-28 04:03:06,951] A new study created in memory with name: no-name-e234e630-fa
3a-495f-b18f-977447c66ae1
[I 2023-10-28 04:04:36,813] Trial 0 finished with value: 0.6265899334185699 and paramete
rs: {'C': 0.009772676530127922, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 0 with
value: 0.6265899334185699.
[I 2023-10-28 04:05:52,117] Trial 1 finished with value: 0.6266377947642323 and paramete
rs: {'C': 5.113356030267681e-05, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 1 wit
h value: 0.6266377947642323.
[I 2023-10-28 04:05:58,225] Trial 2 finished with value: 0.7963696488723357 and paramete
rs: {'C': 0.08951193238410565, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 2
with value: 0.7963696488723357.
[I 2023-10-28 04:05:58,793] Trial 3 finished with value: 0.6273796531387894 and paramete
rs: {'C': 1.5082524345856154e-05, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial
2 with value: 0.7963696488723357.
[I 2023-10-28 04:05:59,416] Trial 4 finished with value: 0.6261591740055832 and paramete
rs: {'C': 0.00012268024715623432, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial
2 with value: 0.7963696488723357.
[I 2023-10-28 04:06:00,789] Trial 5 finished with value: 0.6304428325230141 and paramete
rs: {'C': 8.469490635500838, 'penalty': 'l2', 'solver': 'liblinear'}. Best is trial 2 wi
th value: 0.7963696488723357.
[I 2023-10-28 04:07:26,449] Trial 6 finished with value: 0.6272958988979799 and paramete
rs: {'C': 1.212193956917793e-05, 'penalty': 'l1', 'solver': 'saga'}. Best is trial 2 wit
h value: 0.7963696488723357.
[I 2023-10-28 04:07:27,890] Trial 7 finished with value: 0.6516097026022254 and paramete
rs: {'C': 0.7008653731592931, 'penalty': 'l2', 'solver': 'liblinear'}. Best is trial 2 w
ith value: 0.7963696488723357.
[I 2023-10-28 04:07:29,378] Trial 8 finished with value: 0.6597701352726731 and paramete
rs: {'C': 7.294933832188229, 'penalty': 'l2', 'solver': 'liblinear'}. Best is trial 2 wi
th value: 0.7963696488723357.
[I 2023-10-28 04:08:56,848] Trial 9 finished with value: 0.6265181338832848 and paramete
rs: {'C': 0.00012641955385950287, 'penalty': 'l1', 'solver': 'saga'}. Best is trial 2 wi
th value: 0.7963696488723357.
[I 2023-10-28 04:09:09,853] Trial 10 finished with value: 0.7964773374369897 and paramet
ers: {'C': 86.16052792519358, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:09:22,853] Trial 11 finished with value: 0.7964773374369897 and paramet
ers: {'C': 87.31152262340011, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:09:36,052] Trial 12 finished with value: 0.7964773374369897 and paramet
ers: {'C': 79.29674718687302, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:09:48,738] Trial 13 finished with value: 0.7964534056903311 and paramet
ers: {'C': 14.878460630373661, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:10:01,678] Trial 14 finished with value: 0.7964773374369897 and paramet
ers: {'C': 45.47377119946085, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:10:10,688] Trial 15 finished with value: 0.7964414406760637 and paramet
ers: {'C': 1.2871093025725195, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:10:23,134] Trial 16 finished with value: 0.7964773374369897 and paramet
ers: {'C': 70.11781895853325, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:10:34,674] Trial 17 finished with value: 0.7964534056903311 and paramet
ers: {'C': 2.243610925762075, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
```

```
[I 2023-10-28 04:12:10,367] Trial 18 finished with value: 0.6265899334185699 and paramet
ers: {'C': 84.85717078889846, 'penalty': 'l1', 'solver': 'saga'}. Best is trial 10 with
value: 0.7964773374369897.
[I 2023-10-28 04:12:16,150] Trial 19 finished with value: 0.7964294756617964 and paramet
ers: {'C': 0.16358335374091207, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 1
0 with value: 0.7964773374369897.
[I 2023-10-28 04:12:28,812] Trial 20 finished with value: 0.7964534056903311 and paramet
ers: {'C': 12.706545482290613, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:12:41,931] Trial 21 finished with value: 0.7964773374369897 and paramet
ers: {'C': 96.58105390641938, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:12:55,202] Trial 22 finished with value: 0.7964653715636603 and paramet
ers: {'C': 23.139525422507127, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:13:07,854] Trial 23 finished with value: 0.7964175093589362 and paramet
ers: {'C': 6.958530817568279, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:13:20,828] Trial 24 finished with value: 0.7964773374369897 and paramet
ers: {'C': 32.289421830770756, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:13:32,858] Trial 25 finished with value: 0.7964175093589362 and paramet
ers: {'C': 2.806859476547672, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:14:49,456] Trial 26 finished with value: 0.6265899334185699 and paramet
ers: {'C': 28.225735038275587, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 10 with
value: 0.7964773374369897.
[I 2023-10-28 04:15:01,986] Trial 27 finished with value: 0.7964175093589362 and paramet
ers: {'C': 4.456723638382947, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:15:14,830] Trial 28 finished with value: 0.7964773374369897 and paramet
ers: {'C': 95.70764982054787, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:16:32,045] Trial 29 finished with value: 0.6265899334185699 and paramet
ers: {'C': 0.39505313785665486, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 10 wit
h value: 0.7964773374369897.
[I 2023-10-28 04:16:43,852] Trial 30 finished with value: 0.7964653715636603 and paramet
ers: {'C': 1.3297197274917492, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:16:57,037] Trial 31 finished with value: 0.7964773374369897 and paramet
ers: {'C': 32.916744236118944, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:17:10,037] Trial 32 finished with value: 0.7964773374369897 and paramet
ers: {'C': 35.62016540735398, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:17:22,944] Trial 33 finished with value: 0.7964534056903311 and paramet
ers: {'C': 12.65205037829669, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:17:35,616] Trial 34 finished with value: 0.7964175093589362 and paramet
ers: {'C': 4.753029897500005, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:17:48,367] Trial 35 finished with value: 0.7964773374369897 and paramet
ers: {'C': 98.89117421627871, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:19:09,371] Trial 36 finished with value: 0.6265899334185699 and paramet
ers: {'C': 34.76590981170288, 'penalty': 'l2', 'solver': 'saga'}. Best is trial 10 with
value: 0.7964773374369897.
[I 2023-10-28 04:19:22,613] Trial 37 finished with value: 0.7964534056903311 and paramet
ers: {'C': 13.221017003930822, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
```

```
[I 2023-10-28 04:19:28,263] Trial 38 finished with value: 0.7959269188611792 and paramet
ers: {'C': 0.0456984720330196, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:19:29,622] Trial 39 finished with value: 0.6304428325230141 and paramet
ers: {'C': 4.559636985776951, 'penalty': 'l2', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:21:03,987] Trial 40 finished with value: 0.6265899334185699 and paramet
ers: {'C': 44.74698278188153, 'penalty': 'l1', 'solver': 'saga'}. Best is trial 10 with
value: 0.7964773374369897.
[I 2023-10-28 04:21:17,181] Trial 41 finished with value: 0.7964773374369897 and paramet
ers: {'C': 63.06189616011684, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:21:30,070] Trial 42 finished with value: 0.7964534061198622 and paramet
ers: {'C': 17.475446907325566, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:21:43,176] Trial 43 finished with value: 0.7964773374369897 and paramet
ers: {'C': 54.55700465951723, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:21:56,061] Trial 44 finished with value: 0.7964653711341293 and paramet
ers: {'C': 9.512379909219929, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:22:09,425] Trial 45 finished with value: 0.7964773374369897 and paramet
ers: {'C': 50.23953931387919, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:22:22,164] Trial 46 finished with value: 0.7964534061198622 and paramet
ers: {'C': 17.664407838384324, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:22:23,650] Trial 47 finished with value: 0.6597701352726731 and paramet
ers: {'C': 94.45845863844467, 'penalty': 'l2', 'solver': 'liblinear'}. Best is trial 10
with value: 0.7964773374369897.
[I 2023-10-28 04:22:28,157] Trial 48 finished with value: 0.794012405329885 and paramete
rs: {'C': 0.0072593294053030204, 'penalty': 'l1', 'solver': 'liblinear'}. Best is trial
10 with value: 0.7964773374369897.
[I 2023-10-28 04:24:10,939] Trial 49 finished with value: 0.6265899334185699 and paramet
ers: {'C': 20.871342635705318, 'penalty': 'l1', 'solver': 'saga'}. Best is trial 10 with
value: 0.7964773374369897.
```

In [70]:

```
# Output the Results
for variation_result in results:
    print(f"Variation {variation_result['Variation']}")
    print(f"Best Parameters: {variation_result['Best Parameters']}")
    print(f"Best Cross-Validation Score: {variation_result['Best Cross-Validation Score
```

```
Variation 1
Best Parameters: {'C': 0.5083855249228197, 'penalty': 'l1', 'solver': 'liblinear'}
Best Cross-Validation Score: 0.7964773400141754

Variation 2
Best Parameters: {'C': 0.5894325708616208, 'penalty': 'l1', 'solver': 'liblinear'}
Best Cross-Validation Score: 0.7965132372046323

Variation 3
Best Parameters: {'C': 86.16052792519358, 'penalty': 'l1', 'solver': 'liblinear'}
Best Cross-Validation Score: 0.7964773374369897
```

In [71]:

```
# Train and Evaluate Final Models
for i, variation_result in enumerate(results):
    lr = LogisticRegression(**variation_result['Best Parameters'], max_iter=5000, rando
```

```
    lr.fit(xtrain, ytrain)
    score = lr.score(xtest, ytest)
    print(f"Accuracy of Variation {i + 1} on Validation Set: {score * 100:.2f}%\n")
```

```
Accuracy of Variation 1 on Validation Set: 79.62%

Accuracy of Variation 2 on Validation Set: 79.62%

Accuracy of Variation 3 on Validation Set: 79.62%
```

Week8 (Random Forest)

In [72]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score

base_rfc = RandomForestClassifier(random_state=90)
base_rfc.fit(xtrain, ytrain)
base_pred = base_rfc.predict(xtest)
base_auc = roc_auc_score(ytest, base_pred)
```

In [73]:

```
# Variation 1: Small number of trees, unlimited depth
rfc1 = RandomForestClassifier(n_estimators=10, max_depth=None, random_state=90)
rfc1.fit(xtrain, ytrain)
pred1 = rfc1.predict(xtest)
auc1 = roc_auc_score(ytest, pred1)

# Variation 2: Moderate number of trees, depth of 10
rfc2 = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=90)
rfc2.fit(xtrain, ytrain)
pred2 = rfc2.predict(xtest)
auc2 = roc_auc_score(ytest, pred2)

# Variation 3: Large number of trees, depth of 3
rfc3 = RandomForestClassifier(n_estimators=500, max_depth=3, random_state=90)
rfc3.fit(xtrain, ytrain)
pred3 = rfc3.predict(xtest)
auc3 = roc_auc_score(ytest, pred3)
```

In [74]:

```
print(f"Base Model AUC: {base_auc}")
print(f"Variation 1 AUC: {auc1}")
print(f"Variation 2 AUC: {auc2}")
print(f"Variation 3 AUC: {auc3}")
```

```
Base Model AUC: 0.854700691928137
Variation 1 AUC: 0.8317632978170258
Variation 2 AUC: 0.8141940904686819
Variation 3 AUC: 0.6835733164827276
```

In [75]:

```
import pandas as pd

data = {
    'Variation': ['Base', 'Variation 1', 'Variation 2', 'Variation 3'],
    'n_estimators': [100, 10, 100, 500], # default is 100
    'max_depth': [None, None, 10, 3],
```

```
    'AUC': [base_auc, auc1, auc2, auc3]
}

df = pd.DataFrame(data)
print(df)
```

```
     Variation  n_estimators  max_depth       AUC
0         Base           100        NaN  0.854701
1  Variation 1            10        NaN  0.831763
2  Variation 2           100       10.0  0.814194
3  Variation 3           500        3.0  0.683573
```

In [76]:

```
best_auc = max([base_auc, auc1, auc2, auc3])
best_model = ['Base', 'Variation 1', 'Variation 2', 'Variation 3'][[base_auc, auc1, auc

print(f"The best model is {best_model} with an AUC of {best_auc}")
```

```
The best model is Base with an AUC of 0.854700691928137
```

In [82]:

```
# Calculate training AUCs
base_auc_train = roc_auc_score(ytrain, base_rfc.predict(xtrain))
auc1_train = roc_auc_score(ytrain, rfc1.predict(xtrain))
auc2_train = roc_auc_score(ytrain, rfc2.predict(xtrain))
auc3_train = roc_auc_score(ytrain, rfc3.predict(xtrain))

# Add training AUCs to the dataframe
df['AUC_Train'] = [base_auc_train, auc1_train, auc2_train, auc3_train]
print(df)
```

```
     Variation  n_estimators  max_depth       AUC  AUC_Train
0         Base           100        NaN  0.854701   1.000000
1  Variation 1            10        NaN  0.831763   0.989448
2  Variation 2           100       10.0  0.814194   0.817013
3  Variation 3           500        3.0  0.683573   0.685281
```